《电脑报》跟我学系列丛书

跟我学 Quick BASIC

312
D61

主编：陈宗周
编著：刘秉刚

云南大学出版社

跟我学 Quick BASIC

빠른 QBASIC

Quick QBASIC

TP312
LPG/1

《电脑报》跟我学系列丛书

跟我学 Quick BASIC

陈宗周　主编

刘秉刚　编著

云南大学出版社

内容简介

本书主要介绍 Quick BASIC 这种功能强大、用户众多且又简单易学的编程语言。全书从它的工作环境和语言环境两个方面进行叙述，前者包括编辑器、调试器、库管理工具、编译连接及运行，后者包括语言基础和程序设计，全书安排循序渐进，由浅入深，在书中安排了大量的实例以助广大读者快速掌握这门语言。

本书是面向初学者的一本自学参考书籍。

JS266/11

跟我学 Quick BASIC

编　　者：刘秉刚
责任编辑：张世鸾　李继毛　廖国富
封面设计：李光宇
出　　版：云南大学出版社出版
排　　版：电脑报社照排部
印　　刷：重庆电力印刷厂
经　　销：全国各地新华书店
开　　本：787×1092 1/16
印　　张：11.75　　字　　数：271.5 千字
版　　次：1996 年 12 月第一版第一次印刷
书　　号：ISBN7－81025－730－7/TP・32

定价：12.00 元

序

在人类即将跨入21世纪的今天，计算机技术以前所未有的速度在全世界普及，这是全球范围的科技新潮。50年前，当第一台计算机ENIAC诞生时，被称为“计算机之父”的数学家冯·诺依曼曾预测：“全世界有四台象ENIAC这样的计算机就足够用了”。但事实上，目前全世界的计算机已超过4亿台，而且还以每年新增1亿台的速度在发展。著名科学家在预测上出现如此大的失误，正说明计算机技术的发展远远超过人们的预料。现在计算机已经渗透进人类社会生活的各个领域，发挥着越来越重要的影响。

计算机在全球的普及，反过来对人类提出了新的要求。学习和掌握计算机知识，成了现代人类的新的需求。计算机已经象自然语言和数学一样，成了人类第三种必须掌握的工具。在今天的社会生活中，不懂计算机知识的人已经被称为“新文盲”。

全球计算机普及的热潮很自然地席卷了中国。随着中国经济的腾飞，全国广大群众渴望学习计算机知识、掌握计算机技术，这是中国正在发生的深刻变化。

要学习和掌握一门技术，需要有一个好的向导——计算机普及读物。致力于在中国普及计算机知识的《电脑报》，针对群众学习计算机知识的热潮，组织了计算机教育界的专家，精心撰写了这套适合广大群众自学计算机知识、掌握计算机技术的丛书——“跟我学”电脑系列丛书。

高质量的计算机普及读物应在实用性和通俗性上胜人一筹。它应该使复杂的问题变简单，使高深的专业术语通俗化。这样才能为广大群众接受。我们的计算机科普作家们在这套丛书中正是遵循这一原则，以简单、清晰的语言向广大计算机初学者及初、中级应用人员介绍计算机知识。

从内容选取上，丛书的作者也下了很大功夫。这套丛书内容涵盖面广，凡是PC机使用者在90年代中期可能遇到的软、硬件知识，都精心地编入丛书。大大增强了这套普及读物的实用性，使它既是入门向导，又是入门后的手册和参考资料。

计算机科学是实践性很强的学科，特别注重动手实践。这套书在内容中贯穿了许多操作实例，引导读者由浅入深、循序渐进地掌握对计算机的使用。希望读者勤于上机实践，加深对每一知识点的理解，从而提高计算机应用水平。

计算机充满乐趣，当你亲自动手掌握了这门技术时，你会感到极大的满足和欢乐。

愿这套“跟我学”丛书带你走进计算机世界！

陈省身

1996年12月

关于本书

欢迎您使用“跟我学”电脑系列丛书的《跟我学 Quick BASIC》。

BASIC 是面向计算机初学者的一种广为流行、功能强大、易学好用的程序设计语言，它从 MBASIC、GWBASIC 发展到 Quick BASIC 和最新面向对象程序设计的 Visual BASIC，这种发展说明 BASIC 语言受到众多计算机用户的喜爱和支持。具有基础性编程思想和简单编程理念的 BASIC 将带领广大电脑爱好者跨入神奇的程序设计天地，并将永远受到人们的青睐。正如谭浩强教授所说：“BASIC 大旗不倒！”

Quick BASIC 简称 QB，由 BASIC 发展而来。它既保留了早期 BASIC 易学易懂的人机交互方式，又采用了各种高级语言都使用的编译方式，提高了运行速度，有利于软件的开发。此外，它所采用的模块化的结构程序设计方法，不但易于阅读，也便于设计和维护。Quick BASIC 还提供了集成开发环境，采用屏幕菜单技术，集编辑、编译、调试于一身，使得人机会话直观，操作简便。由于它的简单易学和强大功能，对于步入电脑世界的初学者，用它作入门向导，无疑是一种最佳选择。

本书就是为初学者学习 QB 编程技术编写的自学读物。书中用通俗易懂的语言和大量的实例使读者能很快掌握这门编程语言。

·本书的结构

作为初学者的自学教材，本书在内容和结构安排上尽量做到由浅入深、循序渐进、突出重点、联系实际，可操作性强。全书共分两部分：

第一部分

为前六章，以掌握 QB 的集成环境为主线，同时结合介绍 QB 最基本的典型程序，初学者可以在计算机上边学边操作。

第一章的快速入门，花几个小时就可学会最基本的操作：启动系统、输入源程序、执行源程序、保存源程序。

第二章深入介绍 QB 的编辑功能：文本块的移动、复制、删除的命令。

第三章介绍程序的管理，其核心是模块化的程序设计思想，介绍了模块的建立、组装、保存、连接、打印等管理功能。

第四章讨论程序的调试方法，掌握调试工具会给程序的执行带来很大的方便，同时也是深入了解执行过程的有效途径。

第五、六章分别介绍了 QB 环境和 DOS 环境下建立可执行文件的方法，这对进一步作软件开发，了解 QB 的强大功能是必不可少的。

第二部分

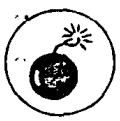
最后一章，全面系统地简介了 QB 的基本语法和常用的语句、函数，供初学者进一步学习，提高编程能力。这一章中精选了大量的程序例子，用来帮助读者理解语句的语法及用途，也是结构化程序设计的示范。

·本书使用的图形符号

为便于您更好地学习本书，书中使用以下三种图形符号，给予您一些有益的提示：

这部分内容较深，你可以跳过去不读

这是对你很有用的经验和小技巧



注意，这是容易出错的地方

目　　录

第一章　快速入门 …… (1)

1.1　电脑与程序 …… (1)
1.2　QB 系统的安装和启动 …… (4)
1.2.1　QB 的运行环境 …… (4)
1.2.2　系统的安装 …… (4)
1.2.3　系统的启动 …… (5)
1.2.4　对显示屏幕的认识 …… (5)
1.3　画面的基本操作 …… (7)
1.3.1　窗口的激活 …… (7)
1.3.2　菜单的打开和退出 …… (8)
1.3.3　简易菜单和完备菜单 …… (9)
1.3.4　File 菜单命令 …… (10)
1.4　源程序的输入与执行 …… (11)
1.4.1　从键盘输入源程序 …… (11)
1.4.2　灵巧编辑器的功能与使用 …… (13)
1.4.3　简单的编辑方法 …… (14)
1.4.4　源程序的执行 …… (15)
1.5　源程序的操作 …… (16)
1.5.1　源程序的保存 …… (16)
1.5.2　源程序的装入 …… (18)

第二章　快捷的编辑 …… (19)

2.1　让电脑重复执行程序 …… (19)
2.1.1　帮国王算清这笔账 …… (19)
2.1.2　电脑如何进行判断 …… (21)
2.2　编辑菜单 Edit 的功能 …… (22)
2.3　文本块的编辑 …… (23)
2.3.1　文本块的选定 …… (23)
2.3.2　文本块的移动 …… (24)
2.3.3　文本块的复制 …… (27)
2.3.4　文本块的删除 …… (29)
2.3.5　文本的查找与替换 …… (29)

第三章　文件的管理 …… (35)

3.1　程序的模块化 …… (35)

3.1.1 如何建立模块 …… (37)
3.1.2 模块的输入 …… (38)
3.1.3 模块的查看 …… (40)
3.2 如何把模块组装成程序 …… (41)
3.2.1 建立主模块 …… (42)
3.2.2 如何调用子程序 …… (45)
3.3 QB 中源文件的类别 …… (47)
3.3.1 模块 …… (47)
3.3.2 包含文件(Include) …… (53)
3.3.3 文档 …… (54)
3.4 文件的合并与打印 …… (55)
3.4.1 文件的合并(Merge) …… (55)
3.4.2 文件的打印(PRINT) …… (56)
第四章 程序的执行与调试 …… (58)
4.1 QB 源程序是怎样被执行的 …… (58)
4.1.1 高级语言与机器语言 …… (58)
4.1.2 QB 的解释功能和编译功能 …… (60)
4.2 如何调试 QB 程序 …… (60)
4.2.1 程序输入时的常见错误 …… (61)
4.2.2 程序执行时检查到的错误 …… (62)
4.2.3 如何查看错误信息 …… (65)
4.3 利用立即窗口进行调试 …… (66)
4.3.1 立即窗口(Immediate)的特点 …… (66)
4.3.2 立即窗口允许的语句 …… (67)
4.4 QB 的调试功能 …… (67)
4.4.1 调试菜单 …… (67)
4.4.2 调试命令的功能 …… (68)
4.5 程序调试实例 …… (71)
4.5.1 了解分支程序的执行过程 …… (71)
4.5.2 一个有溢出(Overflow)错误信息的程序调试 …… (73)
第五章 可执行文件与 QB 库的建立 …… (76)
5.1 建立可执行文件 …… (76)
5.1.1 在 QB 环境建立可执行文件 …… (76)
5.1.2 两种文件生成方式 …… (78)
5.1.3 建立可执行文件所需的程序 …… (79)
5.2 Quick 库 …… (79)
5.2.1 库的分类 …… (79)
5.2.2 Quick 库的优点 …… (80)
5.2.3 Quick 库的建立 …… (80)

5.2.4 如何使用 Quick 库 …… (82)
第六章 在 DOS 环境下建立可执行文件 …… (83)
6.1 编译和连接的一般过程 …… (83)
6.2 用 BC 命令编译 …… (84)
6.2.1 操作过程 …… (84)
6.2.2 BC 命令的选择项 * …… (85)
6.3 用 LINK 命令连接 …… (87)
第七章 用 QB 语言编程 …… (89)
7.1 基本的语法知识 …… (89)
7.1.1 字符集 …… (89)
7.1.2 模块、程序行、语句 …… (89)
7.1.3 数据类型 …… (91)
7.1.4 常量与变量 …… (92)
7.1.5 标准函数 …… (95)
7.1.6 表达式 …… (96)
7.2 输入与输出 …… (99)
7.2.1 赋值语句(LET) …… (99)
7.2.2 变量数值交换语句(SWAP) …… (99)
7.2.3 屏幕显示输出语句(PRINT) …… (100)
7.2.4 键盘输入语句(INPUT) …… (104)
7.2.5 读数、置数语句(READ/DATA) …… (105)
7.2.6 符号常数说明语句(CONST) …… (106)
7.3 选择结构语句 …… (107)
7.3.1 块 IF 语句 …… (107)
7.3.2 行 IF 语句 …… (109)
7.3.3 多分支的块 IF 语句 …… (110)
7.3.4 SELECT CASE 语句 …… (112)
7.4 循环结构语句 …… (112)
7.4.1 计数循环语句(FOR—NEXT) …… (112)
7.4.2 DO—LOOP 条件循环语句 …… (115)
7.5 数组 …… (118)
7.5.1 数组和下标变量 …… (118)
7.5.2 数组和下标变量的表示方法 …… (119)
7.5.3 DIM 语句 …… (120)
7.5.4 一维数组的应用 …… (122)
7.5.5 二维数组的应用 …… (126)
7.6 过程:子程序和函数 …… (127)
7.6.1 FUNCTION 子程序 …… (127)
7.6.2 SUB 子程序 …… (129)

7.6.3 过程中的形实结合 …… (129)
7.6.4 DECLARE 语句 …… (133)
7.6.5 全局量与局部量 …… (133)
7.7 字符串 …… (135)
7.7.1 有关字符串的基本概念 …… (135)
7.7.2 字符串变量的赋值(LINE INPUT 语句) …… (137)
7.7.3 从键盘读入字符(INKEY$ 函数) …… (137)
7.7.4 字符串的连接 …… (138)
7.7.5 字符串的比较 …… (138)
7.7.6 字符串的检索 …… (139)
7.7.7 取子字符串 …… (140)
7.7.8 字符串的生成与字符串中字符的改变 …… (142)
7.7.9 字符串与数的相互转换 …… (142)
7.8 文件 …… (144)
7.8.1 文件的基本概念 …… (144)
7.8.2 用户自定义数据类型 …… (146)
7.8.3 顺序文件 …… (149)
7.8.4 随机文件 …… (153)
7.9 图形 …… (156)
7.9.1 电脑作图的基本知识 …… (156)
7.9.2 有关屏幕控制的语句 …… (158)
7.9.3 画点语句 …… (161)
7.9.4 画线或矩形 …… (162)
7.9.5 画圆 …… (163)
7.9.6 图形的着色 …… (164)
7.10 发声与音乐 …… (165)
7.10.1 发声语句 …… (165)
7.10.2 演奏音乐的 PLAY 语句 …… (167)

附录 A ASC Ⅱ 码表(0～255) …… (172)
附录 B QB 出错信息 …… (174)

第一章

快速入门

本章内容：

- ▶QB 程序的初步认识
- ▶QB 系统的安装、启动、退出
- ▶集成环境画面的初步认识和基本操作方法
- ▶源程序的输入、编辑、执行
- ▶源程序的保存和装入

1.1 电脑与程序

在现代生活中，电脑神话般地在为人们干活，千千万万的人们都为它的魔力所吸引。为什么电脑有这样大的威力？其实，这并不神秘，电脑做的任何一件事情，都要人们事先告诉它如何一步一步地去做，这些电脑能执行的解决问题的步骤就是程序。人们事先没有为它编制好程序，电脑就什么事也干不了。编制程序的工作，就是程序设计，这是很重要的、很有意义的事情。

初学者在没有学习编写程序之前，先看看程序是个什么样子，亲自动手上机去试着执行，得到一些感性认识，对学习是大有好处的，这样还可以逐步熟悉我们的编程环境。

下面将介绍几个用 QB(Quick BASIC 的简写)语言编写的小程序，可能一下子还弄不明白，这不要紧，暂时不去理会它，以后会逐步熟悉的。

例一：在屏幕上显示字符的程序

【源程序】

```
'Demo program of display characters
PRINT "Hello! How are you."
PRINT "Welcome to learn Quick BASIC!"
PRINT "Wish you successful!"
END
```

【执行结果】

这个程序执行后，屏幕上显示以下内容：

Hello! How are you.

Welcome to learn Quick BASIC!

Wish you successful!

【程序解说】

1. 源程序

用高级语言编写的程序，称为源程序(Source Program)，它要用一种叫编译器的程序(Compiler) 翻译成电脑能执行的目标程序(Object Program)，机器才能执行。

2. 程序行

一个源程序是由若干个程序行构成，此程序共有五个程序行。

3. 语句

一个语句一般由两部分组成：语句定义符(又叫语句名)和语句体，如此程序中的PRINT、END 是语句定义符，PRINT 后面的内容是它的语句体，END 语句没有语句体。

4. 注释语句

此程序第一行语句的开始有一单引号“ ’”，表示这是注释语句，该语句的功能是说明程序的内容，便于阅读。

5. PRINT 语句

输出语句，这里是把语句体指定的内容输出到显示屏幕上，语句体中用双引号 " 括起来的一段字符，称为字符串，该语句的功能是把字符串在屏幕上原样显示出来。

6. END 语句

结束语句，程序停止执行。

7. 在汉字操作系统的环境下，例如在 UCDOS 系统下，这个程序还可以用来显示汉字，程序如下：

```
’显示字符的演示程序
PRINT "你好!"
PRINT "欢迎学习 Quick BASIC!"
PRINT "祝你成功!"
END
```

程序执行后，屏幕显示：

你好!

欢迎学习 Quick BASIC!

祝你成功!

例二：唱歌程序

【源程序】

```
' playing a song
PLAY "MNT150L4O3"
PLAY "cdeccdec"
PLAY "efg2efg2"
```

```
PLAY "g8a8g8f8ecg8a8g8f8ec"
PLAY "CO2GO3C2CO2GO3C2"
END
```

【执行结果】

演奏儿歌"两支老虎"

4/4　　两支老虎

1	2	3	1	1	2	3	1
3	4	5	—	3	4	5	—
56	54	3	1	56	54	3	1
1	5	1	—	1	5	1	—

【程序解说】

PLAY 语句：音乐演奏语句，语句体中的字符串是乐曲的代码。

例三：画图程序

【源程序】

```
'drawing a circle
SCREEN 1
CIRCLE (200,200),50
END
```

【执行结果】

屏幕上显示一个圆。

【程序解说】

1. CIRCLE 语句

作图语句，可以画圆、圆弧、椭圆，语句体中的(200,200)指定了圆心的座标位置，50是半径。

2. SCREEN 语句

屏幕定义语句，说明作图的屏幕类型，语法详见第七章。

例四：计算圆面积

【源程序】

```
'Calculating the Circle Area
CONST PI=3.14159
LET R=2.0
LET Area=PI*R*R
PRINT "Area=";Area
END
```

【执行结果】

这个程序计算半径为 2.0 的圆面积，执行程序后，屏幕将显示如下结果：

Area＝12.56636

【程序解说】

1. 简单变量

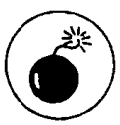
程序中的字符 R、Area 是用来在电脑的内存储器中存放半径和面积的数值的，称为简单变量，简单变量名字以英文字母开头，后跟字母或数字，如 X1、SETA 等等，字符个数不得超过 40 个。

2. LET 语句

赋值语句，用它让电脑把一个变量的值存放到该变量中去。

例如　　R＝2.0　　就是把常量 2.0 存放到变量 R 中去；

　　Area＝PI＊R＊R　它把右边式子计算得到的结果值，存放到变量 Area 中。



此处的“＝”读作赋值号，它与数学上的“＝”(等于号)意义不同。

3. CONST 语句

常量说明语句，把程序中频繁使用的一些常量用符号表示，如圆周率的值 3.14159，用 PI 代替。

1.2 QB 系统的安装和启动

1.2.1 QB 的运行环境

QB(4.5 版)是一个用菜单驱动，集程序的编辑、编译、调试和文件管理于一身的多功能集成编程环境软件。

运行环境：

1. 硬件

IBM－PC/XT、286、386、486、586 及其兼容机上均可运行，内存大于 320KB，至少一个软盘驱动器，最好有硬盘。

如果有一只鼠标器(mouse)的话，操作起来更方便。

2. 软件

DOS 3.0 或以上的版本(或汉字操作系统 UCDOS)。

1.2.2 系统的安装

1. 在硬盘上安装

在有硬盘的电脑上，运行第 1 号软盘上的 SETUP 文件，根据屏幕提示，一步一步地把系统安装到硬盘上去。安装好以后，每次从硬盘上启动 QB 就行了。

2. 无硬盘的安装

在只有一个软盘驱动器而无硬盘的电脑上，不能按上述方法安装系统，只要在软盘上复制一个 QB. EXE 文件，就可以运行 QB。该文件大约占 280KB 的存储空间。用一张 360KB 的低密盘就可装下，但是建议你用 1.2MB 的高密盘，会给你带来更多的方便。例如，可以在这张高密盘上装上操作系统、常用的工具程序，自己编写的源程序也可存放在上面。

1.2.3　系统的启动

1. 启动操作系统 DOS，在 DOS 的提示符下键入 QB，按回车键即可。例如在用软盘启动时，将 QB 系统的软盘插入 A 驱动器中，再键入：

A>QB ＜回车＞

这样，就进入了第一个菜单选择的初始画面，如图 1—1 所示。

2. 如果 QB 系统是建立在硬盘上的一个子目录内，例如在 C 盘的子目录 QB45 上的，操作必须在进入子目录后键入：

C:\QB45>QB ＜回车＞

3. QB 还允许在 QB 命令之后，加入一些参数选择项，各参数的内容，参见有关使用手册。

1.2.4　对显示屏幕的认识

系统启动后，屏幕上出现如图 1—1 所示画面。我们就从这里开始认识 QB 的集成编程环境。

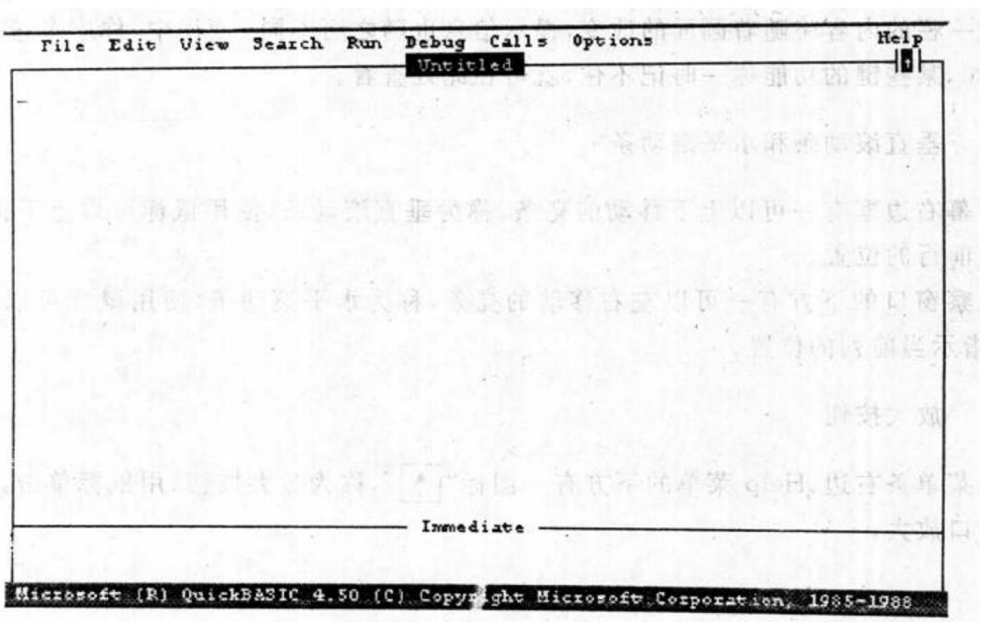


图 1—1　QB 的初始画面

画面可分为四个区域：

□ 菜单条(menu)

在屏幕最上方的一行。其中有九个菜单项，每一项的第一个字母是大写的，且有不同的颜色(在黑白显示器上为高亮度)，我们把它称为主菜单。每一个菜单项下还有下一级的菜单，称为下拉菜单。

□ 观察窗口(View Area)

在菜单条下方的一大块显示区域。我们就在此窗口进行源程序的输入、修改、查看等工作。

窗口的上方有一个标题栏，启动系统后，或已输入了一个新的程序但还未把它作为一个文件存入磁盘，标题栏显示“Untitled”(即没有标题)。一旦你将源程序文件存入磁盘，或将磁盘上的文件调出来，标题栏将显示你的源程序的文件名。

□ 立即窗口(Immediate Area)

观察窗口的下方，标有“Immediate”的一段区域。在这个窗口内，QB 语句可以直接执行。

□ 提示栏(Information)

屏幕最下方的一行。它用英文说明各种功能键的使用方法。

图 1—1 的提示栏内，说明了这个软件的版权、商标和生产日期。

这一栏的内容将随着画面的改变，提示信息也随之而不同。使用中，你应多 多地利用它，比如，某些键的功能你一时记不住，就可在此处查看。

□ 垂直滚动条和水平滚动条

屏幕右边框有一可以上下移动的亮条，称为垂直滚动条，使用鼠标可以上下拖动，它指示当前行的位置。

观察窗口的下方有一可以左右移动的亮条，称为水平滚动条，使用鼠标可以左右拖动，它指示当前列的位置。

□ 放大按钮

在菜单条右边，Help 菜单的下方有一图标“↑”，称为放大按钮，用鼠标单击，可以把观察窗口放大。

1.3 画面的基本操作

1.3.1 窗口的激活

● 工作画面

系统启动后出现初始画面，再按 Enter 键(即回车键)，或按其它键，画面变为图 1—2 所示的工作画面。

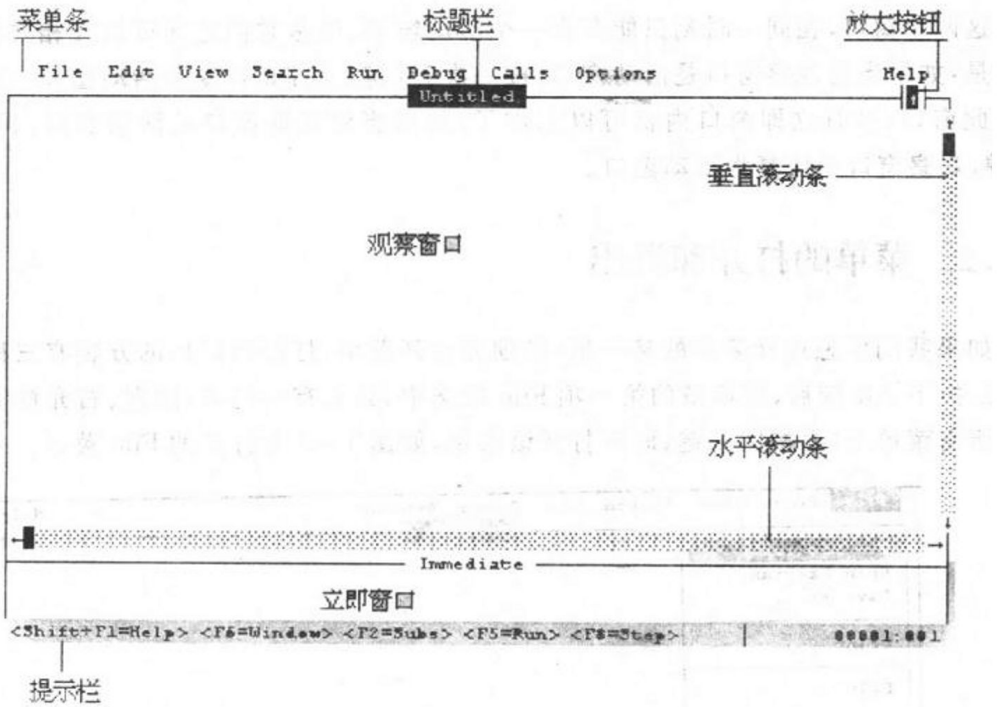


图 1—2　QB 的工作画面

图 1—2 和图 1—1 不同的地方只有提示栏的内容变了，在图 1—2 的提示栏内，告诉你某些功能键的专用功能：

Shift＋F1＝Help　同时按 Shift 和 F1 键进入联机求助系统，可以得到有关使用 QB 的知识。

只有一张低密软盘的系统，有时没有装入这个功能的文件，就不能使用此功能；如果使用了，会在屏幕上显示出错信息。

F6＝Window　用它可以改变活动窗口。

F2＝Subs　进入子程序。

F5=Run	运行 QB 源程序。
F8=Step	调试源程序时，进行单步执行。
N	表示键盘上的 Num Lock 键已按下。
00001:00	表示当前光标所在位置的行和列。它的值将随光标在窗口中的移动而变化。

● 活动窗口

图 1—1 中有两个窗口，工作时只能有一个窗口是可以工作的，光标所在的窗口，称为活动窗口。系统启动后，观察窗口就是活动窗口。立即窗口是不能工作的，称为未激活窗口。

这两个窗口，在同一时刻只能存在一个活动窗口，但是它们之间可以互相转换。转换方法是：如果当前观察窗口是活动窗口，按一次 F6 键，立即窗口变为活动窗口，光标也移到了此窗口，这时立即窗口内就可以工作了；如果当前立即窗口是活动窗口，再 按一次 F6 键，观察窗口又恢复为活动窗口。

1.3.2 菜单的打开和退出

如果我们要想选择菜单的某一项，必须先打开菜单，打开和退出的方法有三种：

1. 按下 Alt 键后，菜单条的第一项 File 被选中，其上有一亮条，用左、右光标键移动亮条到所需菜单上，再按回车键，即可打开该菜单，如图 1—3 为打开的 File 菜单。

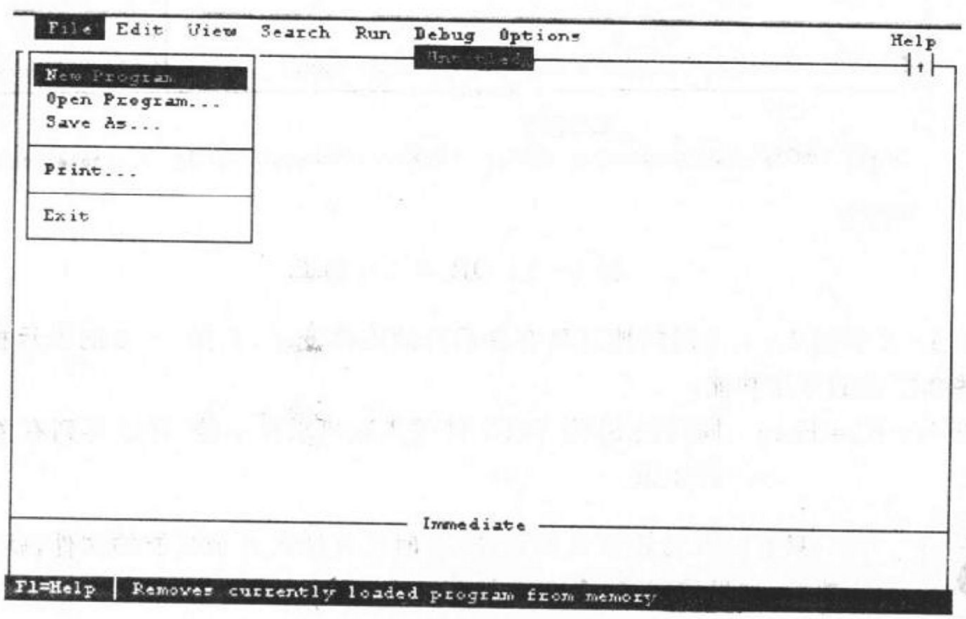


图 1—3　File 的下拉菜单(简易菜单)

Alt 键相当于一个触发开关，按 Alt，打开菜单条，再按 Alt，又退出菜单条，每次是交

替变换的。

这时，按一次 ESC 键，又退出菜单条。

2. 键入 Alt＋菜单项的第一个大写字母，例如按 Alt＋F 键，就选中 File，按 Alt＋E 键就选中 Edit 等等。这时就进入该菜单的下拉菜单。

在菜单选中的状态下，按 ESC 键，退出菜单的选择。

3. 在安装了鼠标器的系统上，直接用鼠标单击要选择的菜单即可。

1.3.3　简易菜单和完备菜单

● 什么是简易菜单和完备菜单

QB 的每一菜单项的下拉菜单都有简易菜单和完备菜单：

简易菜单——提供常用的、主要的操作命令，适用于初学者。如图 1－3 就是 File 的简易菜单。

完备菜单——在简易菜单的基础上增加了更多的操作命令，可以满足编程人员的需要。如图 1－4 就是 File 的完备菜单。

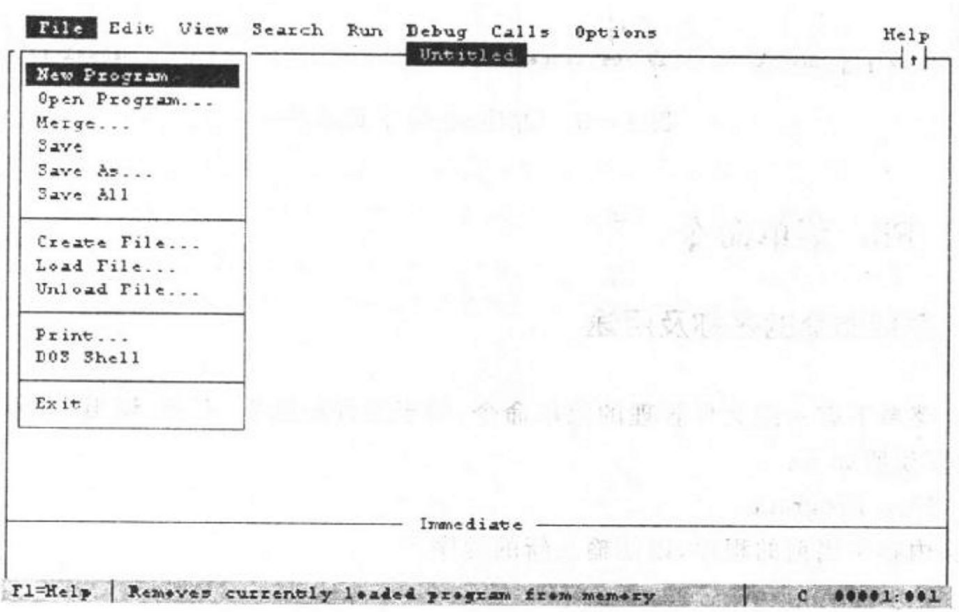


图 1－4　File 的下拉菜单(完备菜单)

● 如何选择简易菜单和完备菜单

1. 用 Alt 键，进入菜单带，选 Options，按回车键，出现图 1－5；

2. 用↑、↓键将光标移到 Full Menu，这时要看该命令前有无表示选中的符号“·”，如果无选中的符号“·”，按回车键，菜单就从当前的简易菜单转到完备菜单，在该项前出

,现选中的符号“·”;

如果已经有选中的符号“·”,按回车键,菜单就从当前的完备菜单转到简易菜单,在该项前的选中的符号“·”消失。

由此可见,选择可以交替进行。

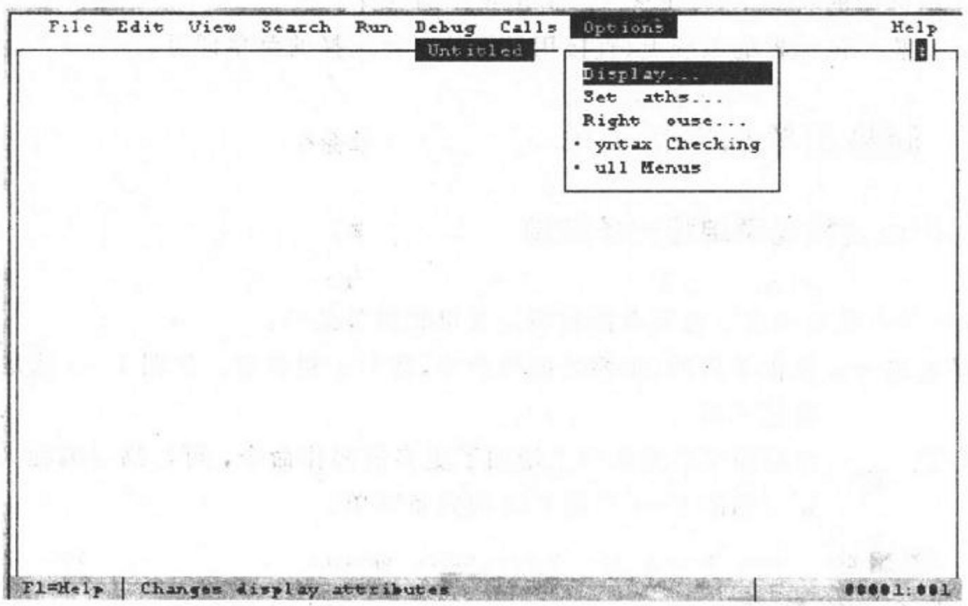


图 1—5 Options 的下拉菜单

1.3.4 File 菜单命令

● 菜单命令的名称及用途

File 菜单下有一组文件管理的菜单命令,用于文件的建立、存储、调出、打印,它们的功能,简要说明如下:

□ New Program

清除内存中当前的程序,以便输入新的程序。

□ Open Program …

把磁盘上的程序装入内存,并在屏幕上显示程序清单。

□ Save As …

将当前内存中的程序,作为一个新文件存放到磁盘上。

□ Print …

在打印机上输出文件。

□ Exit

退出 QB,返回到 DOS 状态。

有的命令后带一个省略号"…",表示选择该命令后,屏幕上将还有一个对话框出现,详见以后各节。

● 选择菜单命令

打开菜单后,可用下列方法选择菜单命令:

1. 第一项命令上有黑色亮条,表示当前要选的菜单命令,可以用上下光标键移动亮条到你所需要的任一项菜单命令,再按回车键。

2. 菜单命令的字符中有一个高亮度字母,键入该字母,就可选择相应菜单项。例如N,就代表了New Program,只要键入N,就选择了New Program,键入A,就选择了Save As …。

3. 用鼠标单击菜单命令即可。

1.4 源程序的输入与执行

1.4.1 从键盘输入源程序

电脑执行的程序必须事先存放在内存中,因此,我们必须先把编好的程序输入到内存中去。一般有两种方法:一种方法是把一个编写好的新程序,从键盘上敲进去;另一种方法是把存放在外存储器(如磁盘、光盘等)上的文件装入到内存中(又称为读文件到内存)。

● 直接从初始画面键入

系统启动后,内存中无程序,屏幕上标题栏显示"Untitled",光标位于第1行第1列,这时可以直接键入一个源程序。输入方法就是按照源程序的内容,从第一行开始(光标位置处在第1行第1列),逐个字符去敲键盘上相应的键,从左至右,光标随之右移,直到该行字符敲完,每敲一个字符,屏幕上都将显示其内容。敲完一行,按一次回车键(即Enter键),光标自动换行到下一行的第1列的位置。这样一行一行地直到全部程序输入完毕,源程序就被输入到内存中了。

输入的程序行,又称文本行,用下列方法之一,均可存入内存:

1. 按回车键(即Enter键)
2. 用光标移动键把光标移开当前行
3. 用鼠标器时,将光标拖离当前行

● 选菜单命令"New Program"输入源程序

如果不是刚启动系统，在操作过程中，内存已有程序驻留，例如，已执行完某个程序，接着还要输入第二个程序的时候，必须用 New Program 命令去清除内存。

执行 New Program 命令后屏幕上会出现一个对话框，如图 1—6 所示。只有在系统启动后，还没有输入或装入程序的时候，执行 New Program 命令，才不出现对话框。

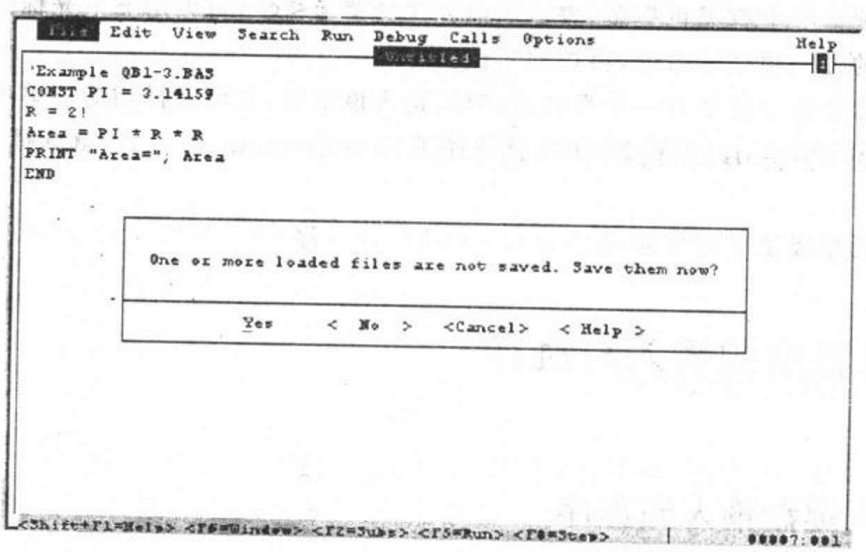


图 1—6　New Program 的对话框

对话框的内容是提示你：在内存中驻留的程序还没有存入磁盘，你怎么办？

下面有四种选择：

□ <Yes>

进行存盘处理，然后清除内存。

□ <No>

不进行存盘，就清除内存。

□ <Cancel>

不清除内存，New Program 命令无效。

□ <Help>

请求联机求助。

用 TAB 键把光标移动到其中之一，按回车键即选择这一项。

请你一定读懂这个提示信息，电脑很友好地帮助你，不要由于一时的粗心，把内存中还没有存盘的程序清除了。因此，该对话框出现后，你还来得及改正，当确认以后，选<yes>即可，这样内存中还没有存盘的程序就进行存盘处理，然后清除内存。

如果真的是不想存盘，那么就选择<No>，源程序就从内存中消失了，如果是一个新输入的程序，因为没有存盘，以后就得重新输入，所以一定要小

心。

如果你不是输入的新程序，而是把存在盘上的程序装入到内存的，选<Yes>，将把内存的程序再一次覆盖同文件名的程序，例如，对程序已作的某些修改，那么，最新的内容将替换原有的内容，确认不需要存盘，才选<No>。只有选择了<Yes>或<No>，内存被清除后，才能进行新程序的输入。

如选择了<Yes>，还会出现下一个对话框，进行存盘处理。

选择<Cancel>，则内存未被清除，不能输入新的程序。当你误动作，不该选 New program，而你把它选了，这时可选<cancel>，退回原来的状态，内存的程序不变。

1.4.2　灵巧编辑器的功能与使用

QB 为用户提供了一个灵巧编辑器，使编辑源程序变得很容易，它具有如下两大功能：

● 自动进行语法检查

当你输入完一行程序，按了回车键后，系统就会检查这一行有无语法错误，如果有错，就会在屏幕上出现一个对话框，如图 1—7 所示，显示错误信息。

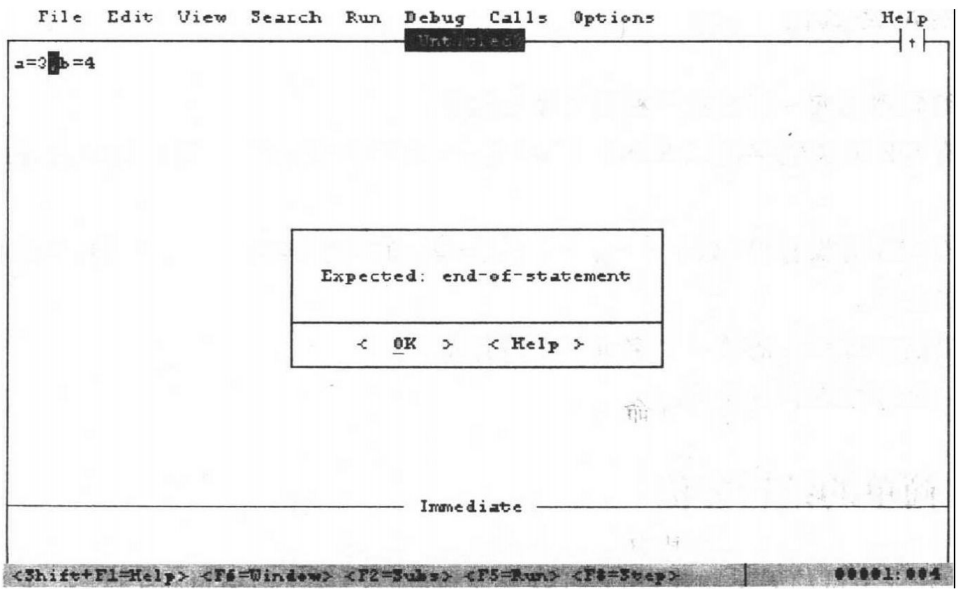
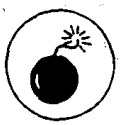


图 1—7　出错信息的对话框

对话框内的英文，表示有语法方面的错误。下方的<Ok> <Help>是处理错误的选择，<Ok>表示你已经知道是什么错误，下一步去改正错误；<Help>表示当你不明白是什么错误时，可进一步查看联机帮助信息。

当看到此对话框后，可按 Esc(或 Spacebar)键清除，清除对话框后光标会置于出错的语句行，且该行亮度增强。

没有错误的语句行，编辑器就把这一行翻译成可执行的机器代码。



有一些输入错误，灵巧编辑器不能识别，要到程序执行时才能发现。例如，输入如下的输出语句：

PRINT X,Y

错误的敲成 PINRT X,Y

这时，系统把 PINRT 当作一个合法的 SUB 子程序名而不出现出错信息的对话框，到了执行该程序时，他找不到这个子程序，才报告出错信息。

系统启动后语法检查就自动开启。

可使用以下方法开启与关闭语法检查：

选“Option”菜单，该菜单中有一命令“Syntax Checking”，其功能就是语法检查的开启与关闭，当语法检查是开启的时候，该命令行前有一个符号“·”，表示是开启状态，当语法检查是的关闭的时候，该命令行前没有符号“·”，表示是关闭状态；将光标移动到该命令行，按回车键，状态会交替变化，符号“·”也随之有无，就是说：当前是开启状态的将变为关闭状态，当前是关闭状态的将变为开启状态。

● 自动格式化

灵巧编辑器把每一行按以下标准自动格式化：

1. 所有 QB 的关键字(如 PRINT、END 等)以大写字母出现，不管你敲的大写或小 写字母。

2. 所有 QB 的操作符(如＋、－、＊、/、＝等)的前后各插入一个空格，不管你敲入时加或没加空格。

3. 变量名和过程名的第一个字母用大写打印。

4. 适当地方加进标点符号。

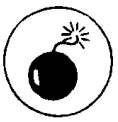
1.4.3 简单的编辑方法

QB 具有很丰富的编辑功能，用它可以方便地进行源程序的编辑和字处理。它有对字符、字的编辑(删除、插入、置换)，有对行和块的编辑，本节只介绍以下简单的编辑方法。

● 删除一个或几个字符

首先把光标移动到要删除的字符下方，按一次 Del 键，该字符被删除，连续按 Del 键，

此字符右边的字符会被连续删去。



删一个字符，按 Del 键的动作要快，如果迟迟不放，就会删去多个字符。

● 插入一个或几个字符

首先将光标移动到要插入的字符位置的右边一个字符上，光标必须是一下横线，这时直接按要插入的字符就行了。例如要在“Welcme”中，字母 c 和 m 之间插入一个字母 o，则将光标移至 m 处，再按字母键 o，字符插在光标左方，“Welcome”就插好了。

光标在“—”状态时，编辑处于 Ins(插入)状态，这时按一次 Ins 键，光标将变为“■”状态，这不是插入状态，而是置换状态了。按 Ins 键，可以交替改变这两种状态。

● 置换一个或几个字符

首先将光标移动到置换字符的位置，按 Ins 键，光标将变为一个方块“■”，再敲入要置换的字符，例如“PRIMT”中的“M”，要用“N”置换，将光标移至“M”且要变为方块，按“N”键，就置换成“PRINT”了。

置换的操作等于先删除后插入两个操作，即先删除不要的字符，再插入要替换的字符。

1.4.4　源程序的执行

执行源程序有以下两种操作方法：

1. 按 Shift＋F5 键。

2. 在 Run 菜单的下拉菜单中，选菜单命令“Start”后，按回车键。

程序执行后的情况：

·执行过程中出现了错误，需要根据出错信息，查找错误原因，排除错误，再次执行程序，直到全部过程不出错误，这就是程序的调试，以后还会介绍一些调试的方法和工具。

·执行过程中如没有错误发现，屏幕将发生变化，从 QB 的菜单画面，回到 DOS 状态的画面(黑底白字的屏幕，如图 1—8 所示)。

```
C:\CL\QB45>qb
Hello! How do you do.
Welcome to learn Quick BASIC.
Wish you successful!
Hello! How do you do.
Welcome to learn Quick BASIC.
Wish you successful!

Press any key to continue
```

图 1—8　执行程序后的 DOS 画面

屏幕下方提示“press any key to continue...”意思是：你按任意一键，就可回到 QB 菜单画面。

3. 从 QB 菜单画面到执行结果的 DOS 画面的变换

按 F4 键或选 View 菜单中的菜单命令“Output screen”。

1.5　源程序的操作

1.5.1　源程序的保存

一个键入的源程序，经过调试、执行后，需要从内存转存到磁盘上，以便日后再用。源程序存盘的方法是在 File 菜单中选“Save As ...”菜单命令，按回车键后，屏幕将出现如图 1—9 所示的对话框。

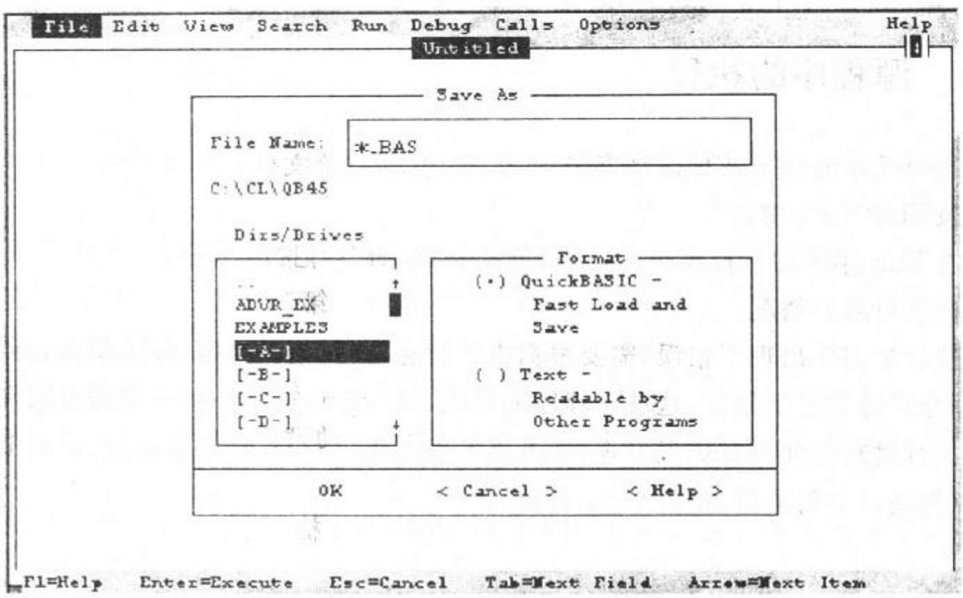


图 1—9　Save As 对话框

在该对话框中有下列内容：

□ File Name

在方框内要输入一个文件名，文件名以英文字母开头后跟英文字母或数字，字符个数不超过8个。

你只要将光标移动到“*.bas”的“*”号左边，插入文件名，删除“*”号即可，扩展名bas不要重输，也不要改变。

QB规定源程序的文件名必须用.bas作扩展名。

□ Dir/Drives

Dir/Drives　用于改变当前驱动器号。

□ Format

指定存放文件的格式，有以下两种存放格式：

Quick BASIC格式——这是QB独有的专用于QB的程序，与其它BASIC不兼容，这种格式存取速度快。

Text格式——文本格式，是以ASCII存放的文件。QB中的文档(Document)文件和包含(Include)文件只能以这种格式存放。

(*)格式选项前的括弧内，有“*”号，表示当前所选的格式，进入此画面时，它在Quick BASIC格式，这是默认的选择，如果要改变的话，将光标移动到下一个格式的括弧内，按回车键。

□ <Ok>、<Cancel>、<Help>

<Ok>　执行存盘

<Cancel>　不执行存盘，退出存盘对话框

<Help>　查看联机求助信息

按TAB键，光标可以在上述各栏循环移动。

存盘操作的过程为：

1. 将光标移动到文件名框内，在“*”号前插入你给文件起的文件名字，最后要删去“*”号，保留扩展名.BAS。

2. 将光标移动到Dirs/Drives栏，再按上下光标键，把亮条移到要存放文件的驱动器号和目录，并按回车键。如果要存放的路径和对话框的第二行显示的相同，可不在此栏选择。

3. 将光标移到Format栏，在两种格式中任选其一。(*)表示已选中的格式，进入对话框，默认选择就是QuickBASIC格式。按光标上下移动键，“*”号就可移动，表示选项改变。

4. 将光标移动到〈Ok〉项按回车键，执行存盘，对话框消去，恢复观察窗口的原貌，且在标题栏内，显示已存盘的文件名。

1.5.2 源程序的装入

把已存放在磁盘上的源程序文件读到计算机的内存中去，以便再次修改、调试或执行。这种操作我们称为源程序的装入。

在 File 菜单中选“Open Program”菜单命令，按回车键后，屏幕出现如图 1—10 所示的对话框。

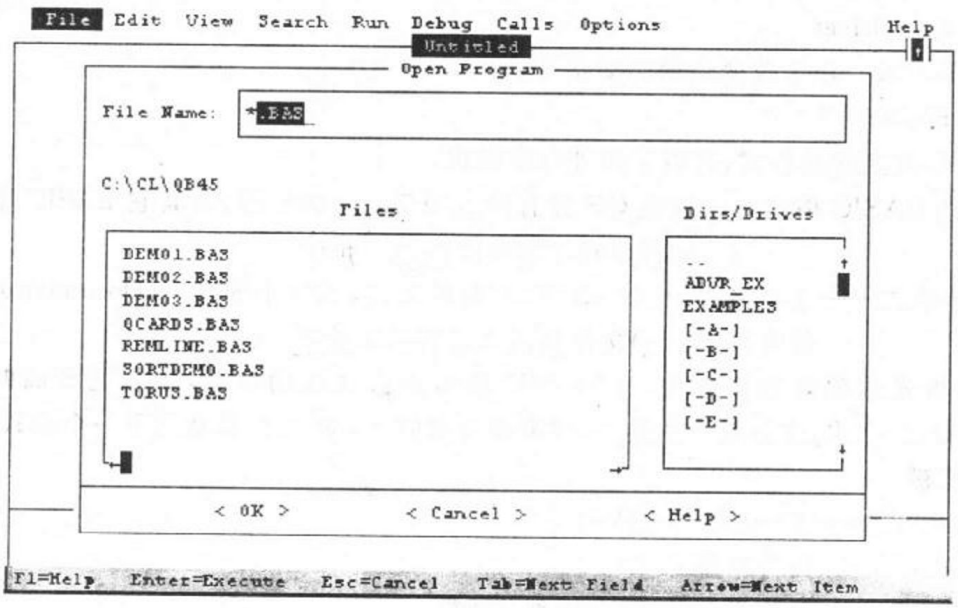


图 1—10　Open Program 对话框

□　File Name

可在此框中输入要装入的文件名。

□　files

在此列表框中显示当前目录下所有以.BAS 为扩展名的文件。

□　Dirs/Drivers

可选择要装入文件所在的磁盘和路径。

装入文件的操作过程为：

1. 按 TAB 键，光标移动到 Files 框上；

2. 再按上下光标移动键，选择要装入的文件，被光标所选的文件名，呈黑白反视(或高亮度条)。

3. 将光标移动到〈Ok〉，按回车键(或只按回车键)

这样，源程序就装入到内存了，对话框消失，观察窗口显示源程序的内容，标题栏内也显示装入的文件名。

只有需要改变当前驱动器号时，才将光标移动到 Dirs/Drives 框内去。先选盘和目录，然后再执行以上操作。

第二章

快捷的编辑

本章内容：

- ▶重复执行程序——循环
- ▶算法的概念和流程图的表示法
- ▶关系表达式的写法
- ▶Edit、Search 两个菜单的功能
- ▶文本块的编辑：如何定义文本块，如何移动块和复制块
- ▶在文本中进行查找和替换

2.1 让电脑重复执行程序

2.1.1 帮国王算清这笔账

传说印度过去有一位名叫舍罕的国王和他的宰相达依尔下棋，宰相很聪明，赢了国王，国王要奖励他，问宰相需要什么，宰相客气地说："国王只要在棋盘的第一个格子里放一粒麦子，第二格放二粒，第三格放四粒，下一格放的粒数是上一格的二倍，一直放到 64 格，我就感恩不尽，其它什么也不要了。"国王想了想：这有多少，还不容易。于是让人扛来一袋小麦，但是，不到一会儿，一袋全完了，再来一袋，很快又完了，一袋又一袋地搬，仓库里的麦子全完了，棋盘还没有放完，结果全印度的粮食拿来都不够，国王纳闷，怎么也算不清这笔账。

现在请你用计算机来帮国王算清这笔胀。

棋盘的示意图如图 2—1。

【算法】

首先，计算总共所需要的麦子的粒数，用公式表示如下：

$$T=1+2+2^2+2^3+\ldots+2^{63}$$

其次，按每一立方米(M^3)有 1.42×10^8 粒，算出总的体积 S

$$S=T/(1.42\times10^8)$$

要让计算机来作这件事，还得更详细的分解成小的步骤，比如计算 T，就得计算 2 的乘方，再求 64 项乘方的和，各乘方项的指数，是有规则的变化的，从 0 逐一增加到 63，我

们可以利用这一规律，让计算机重复做乘法，此外，还要为计算机指定存放这些数值的存储单元，即指定变量，如用 P 存放乘方，T 存放总和，N 存放重复执行的次数。

这些解决问题的步骤，就叫算法。我们常常用流程图的方法来表示，如图 2—2 所示。

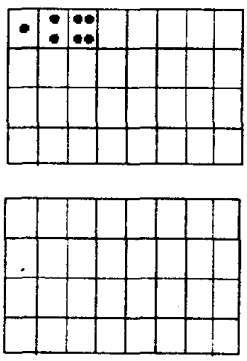


图 2—1　棋盘的示意图

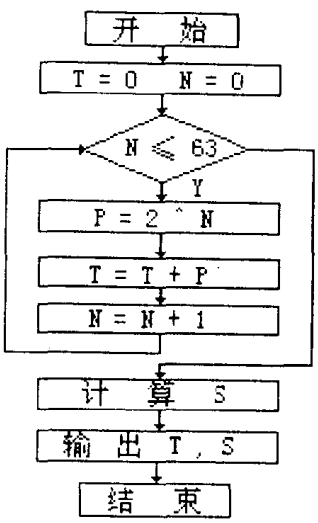
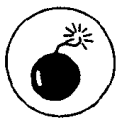


图 2—2　流程图

【源程序】

```
'Example QB2—1.BAS
N=0 :T=0                        '给变量赋初值
    DO WHILE N<=63              '当型循环的条件语句
        P=2^ N                  '计算每项的乘方
        T=T+P                   '计算各项的和
        N=N+1                   '乘方的指数增 1
    LOOP                        '循环结束标志
    S=T/(1.42E+08)              '计算体积
    PRINT "T=";T,"S=";S         '输出结果
    END
```



QB 允许程序行后加写注释，就如上面源程序那样，这可以让人很容易看懂。但是必须在中文操作系统下才能输入汉字，否则，要改为英文或汉语拼音。如果不输入这部分，不会影响程序的执行，因为电脑编译这个程序时，并不编译注释语句。

2.1.2 电脑如何进行判断

电脑不同于其它任何机器的地方就是电脑具有逻辑判断能力，譬如，对两个数进行比较，它会知道哪一个大哪一个小或者相等；它能进行逻辑运算，譬如最基本的逻辑与、逻辑或、逻辑非。

● 条件表达式

要让电脑进行比较，就要告诉谁和谁比较，因此 QB 规定了一套进行关系比较的符号，称为关系运算符，其表示法如表 2—1 所示。

表 2—1　关系运算符

关系运算符	数学运算符	意　义
>	>	大于
<	<	小于
>=	≥	大于等于
<=	≤	小于等于
=	=	等于
<>	≠	不等

在关系运算符的两边写上比较的对象，允许是常量、变量、算术表达式。

例如：N<=63

43<34

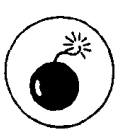
X<=Y

A+B+C>=0

这就是关系表达式，电脑对这样的式子进行运算能得到两种结果：

关系为真，我们称为条件成立；

关系为假，我们称为条件不成立，例如 43<34 的关系就不成立。



关系表达式中的变量必须首先得到具体的值之后才能进行关系运算，例如 N<=63 中当 N 的值从 1 变 63，这个关系表达式的值都为真，如果 N 再加 1，变成 64 了，这时这个关系表达式就不成立了，它的值变为假。

2.2 编辑菜单 Edit 的功能

将光标移动到菜单条的“Edit”，按回车键，出现图 2—3 所示画面。

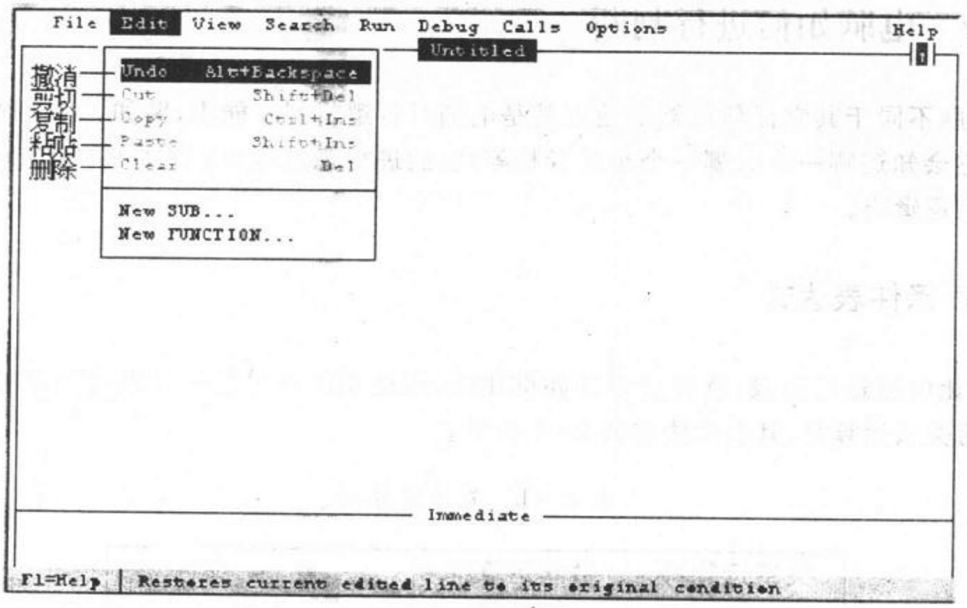


图 2—3　Edit 画面

该菜单中有两组命令：

● 文本编辑命令

□ Undo　撤消

对已作的编辑发现有不当的时候，用该命令可以取消当前所作的编辑。

□ Cut　剪切

把当前选定的文本块剪取下来，送到内存里一个特定区域（称为缓冲区，或称剪切板），剪切后的原文本中就没有这个文本块。

□ Copy　复制

把当前选定的文本块原样复制到缓冲区（或称剪切板），复制后的原文本中仍然保留有这个文本块了。

□ Paste　粘贴

把剪切或复制到缓冲区的文本块，从缓冲区再复制到一个新指定的位置上去。

□ Clear　删除

把当前选定的文本块删除。

● **子程序输入命令**

□ New SUB…
□ New FUNCTION…

在没有选定块之前，命令的颜色均为灰色，表示都处于非活动状态，这时选任何一条命令都无效。

2.3 文本块的编辑

2.3.1 文本块的选定

在一个比较长的程序中常常需要作几行程序的移动、复制、删除等编辑，称之为文本块的操作，利用 Edit 的命令就能很方便地实现。

为了移动、复制、删除一个文本块，首先要作定义文本块的操作，或者又称为文本块的选定。这就是说，无论是移动也好，复制也好，先要告诉电脑操作的对象，否则无法操作，所以，在没有定义块之前，进入 Edit 菜单，编辑命令的字符颜色呈淡灰色而不是黑色，如图 2—3 所示，淡灰色状态称为非活动状态，这时编辑命令不起作用；一旦定义了文本块，编辑命令的字符颜色就由灰变黑，如图 2—4 所示，这种状态称为活动状态，这时编辑命令就可以工作了。

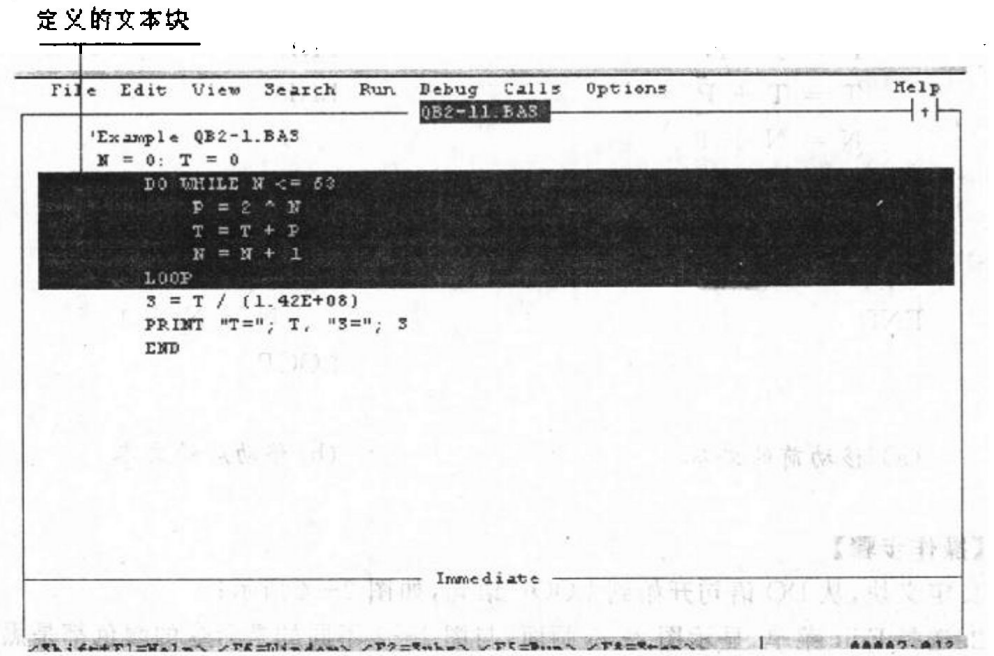


图 2—4　定义文本块

选定文本块有以下两种方法。

1. 使用键盘

(1) 用光标移动键把光标移动到所选文本块的第一行；

(2) 同时按 Shift ＋↓，每按一次，该行变为黑色；

(3) 连续按 Shift ＋↓，直到要选文本块的最末一行，这时要选的文本块全变为黑色，块就定义好了，如图 2－4 所示。

2. 用鼠标器，按住左键不放，从要选文本块的第一行拖到要选的文本块的最末行，要选的文本块全变为黑色，块就定义好了。

2.3.2 文本块的移动

块移动的操作步骤为：

1. 定义文本块；

2. 用 Cut 命令把文本块剪切到缓冲区(剪切板)；

3. 光标移动到指定的目的位置；

4. 用 Paste 命令把文本块从缓冲区移动到目的位置。

例：将 2.1 节的源程序中的 DO 循环移动到程序外，如下所示。

```
'Example QB2－1.BAS
N = 0: T = 0
    DO WHILE N <= 63
        P = 2 ^ N
        T = T + P
        N = N + 1
    LOOP
    S = T / (1.42E+08)
    PRINT "T="; T, "S="; S
    END
```

(a) 移动前的文本

```
'Example QB2－1.BAS
N = 0: T = 0
    S = T / (1.42E+08)
    PRINT "T="; T, "S="; S
    END

    DO WHILE N <= 63
        P = 2 ^ N
        T = T + P
        N = N + 1
    LOOP
```

(b)移动后的文本

【操作步骤】

1. 定义块，从 DO 语句开始到 LOOP 语句，如图 2－4 所示；

2. 选择 Edit 菜单，显示图 2－5 画面，与图 2－3 不同的是命令的颜色都是黑色，都处于活动状态；

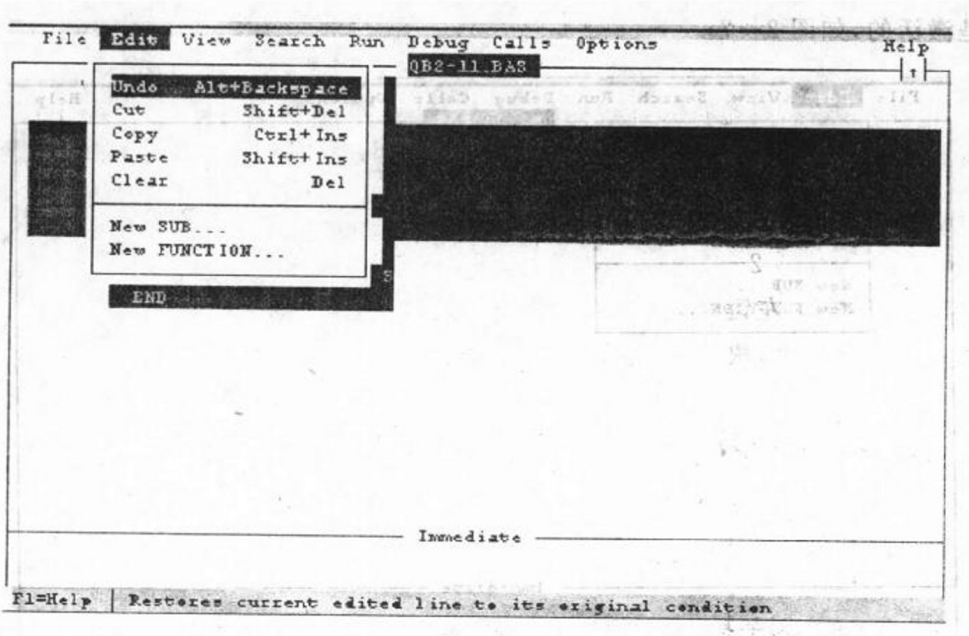


图 2—5　定义了块以后的 Edit 画面

3. 光标移到 Cut 命令，按回车键，或用鼠标单击此命令，画面变为图 2—6；

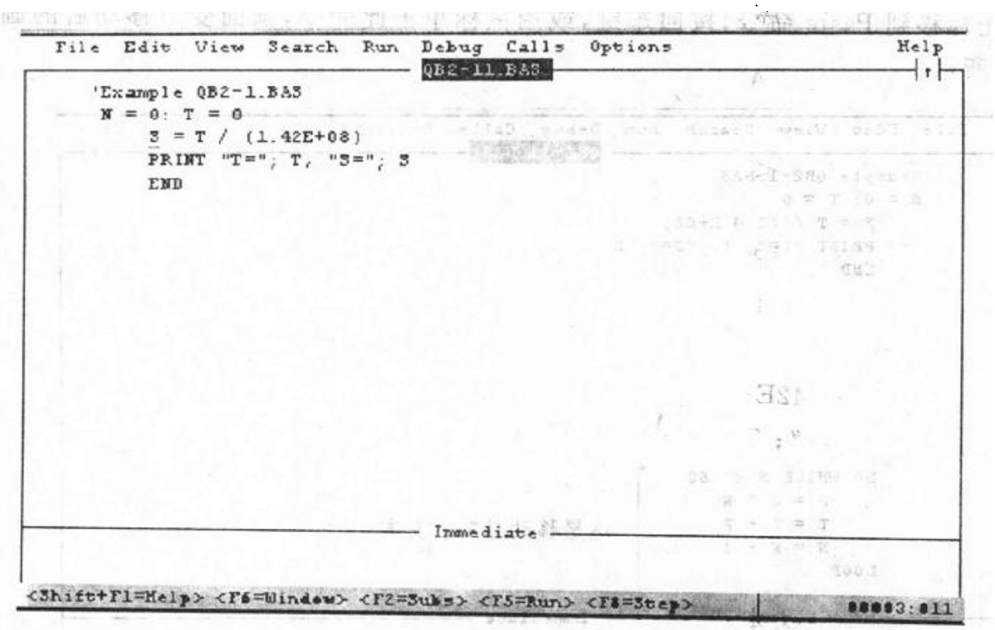


图 2—6　剪切后的画面

定义的文本块已从文本中消失，别担心，不会丢失，它已存放到剪切板中。

4. 光标移动到指定位置，例如可以把它移到当前源程序外的某个地方；

5. 再次进入 Edit 菜单，这时菜单中的命令有的已变为不激活状态，只有 Paste 呈黑

色，才是激活的，如图 2—7；

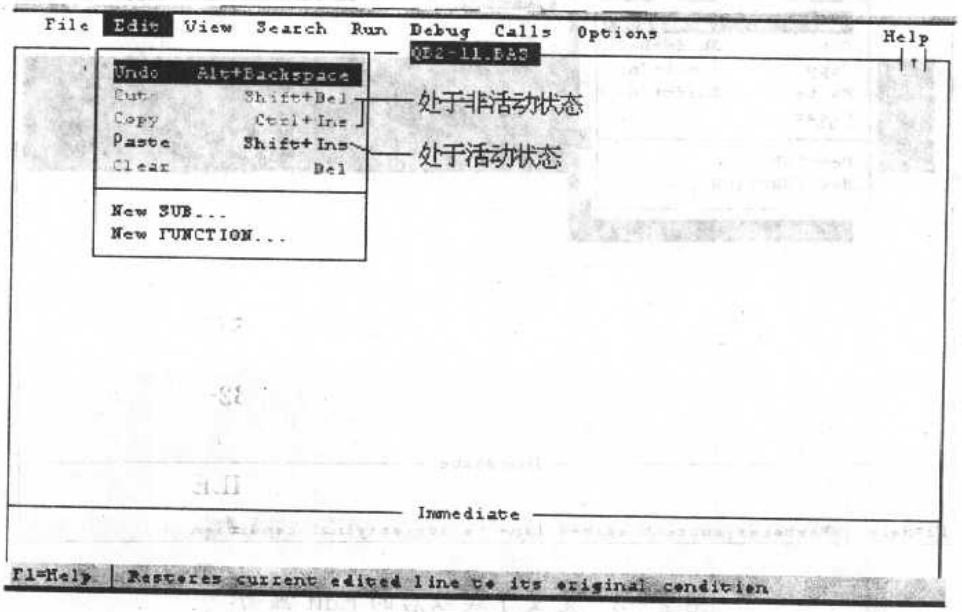


图 2—7　剪切后的菜单状态的变化

6. 光标移到 Paste 命令，按回车键，或用鼠标单击此命令，画面变为移动后的画面如图 2—8 所示。

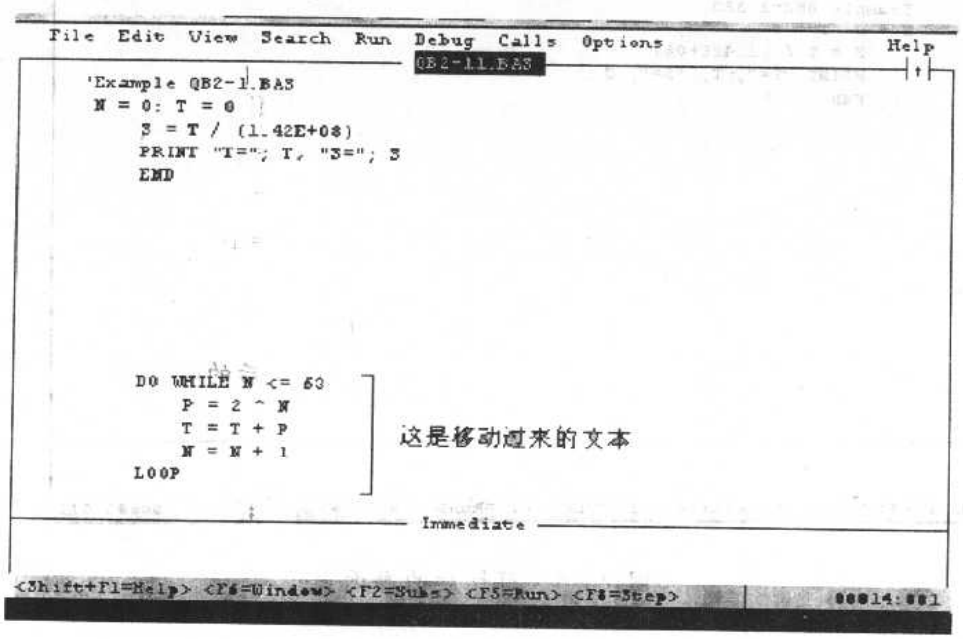


图 2—8　粘贴后的文本

文本块不仅可以在当前文本内移动，而且可以移动到不同的模块或其它文件中去。

2.3.3　文本块的复制

块复制的操作步骤为：

1. 定义文本块；
2. 用 Copy 命令把文本块复制到缓冲区(剪切板)；
3. 光标移动到指定的目的位置；
4. 用 Paste 命令把文本块从缓冲区复制到目的位置。

例：我们将 2.1 节源程序中的 DO 循环复制到程序外，如下所示。

```
'Example QB2－1.BAS
N = 0: T = 0
    DO WHILE N <= 63
        P = 2 ^ N
        T = T + P
        N = N + 1
    LOOP
    S = T / (1.42E+08)
    PRINT "T="; T, "S="; S
    END
```

(a) 复制前的文本

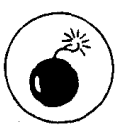
```
'Example QB2－1.BAS
N = 0: T = 0
    DO WHILE N <= 63
        P = 2 ^ N
        T = T + P
        N = N + 1
    LOOP
    S = T / (1.42E+08
    PRINT "T="; T, "S="; S
    END

    DO WHILE N <= 63
        P = 2 ^ N
        T = T + P
        N = N + 1
    LOOP
```

(b)复制后的文本

【操作步骤】

1. 定义块，从 DO 语句开始到 LOOP 语句，如图 2－4 所示；
2. 选择 Edit 菜单，进入图 2－5 画面；
3. 光标移到 Copy 命令，按回车键，或用鼠标单击此命令；



定义的文本块不会从文本中消失，这和 Cut 命令不同。

4. 光标移动到指定位置，例如可以把它移到源程序外；
5. 再次进入 Edit 菜单，菜单中的命令有的已变为不激活状态，只有 Paste 呈黑色，

才是激活的，如图 2—9；

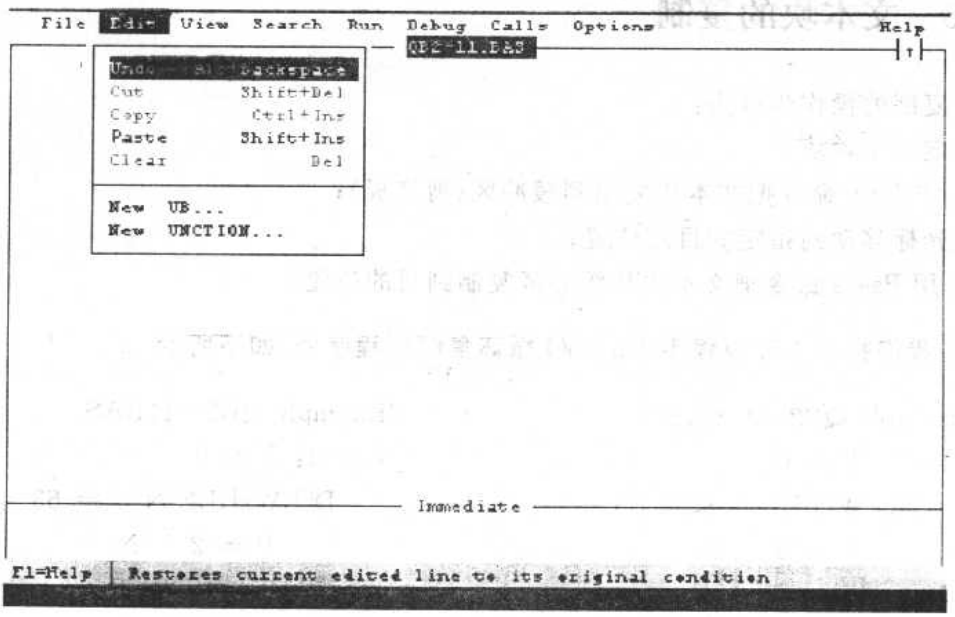


图 2—9　复制后的 Edit 菜单

6. 光标移到 Paste 命令，按回车键，或用鼠标单击此命令，画面变为复制后的画面如图 2—10 所示。

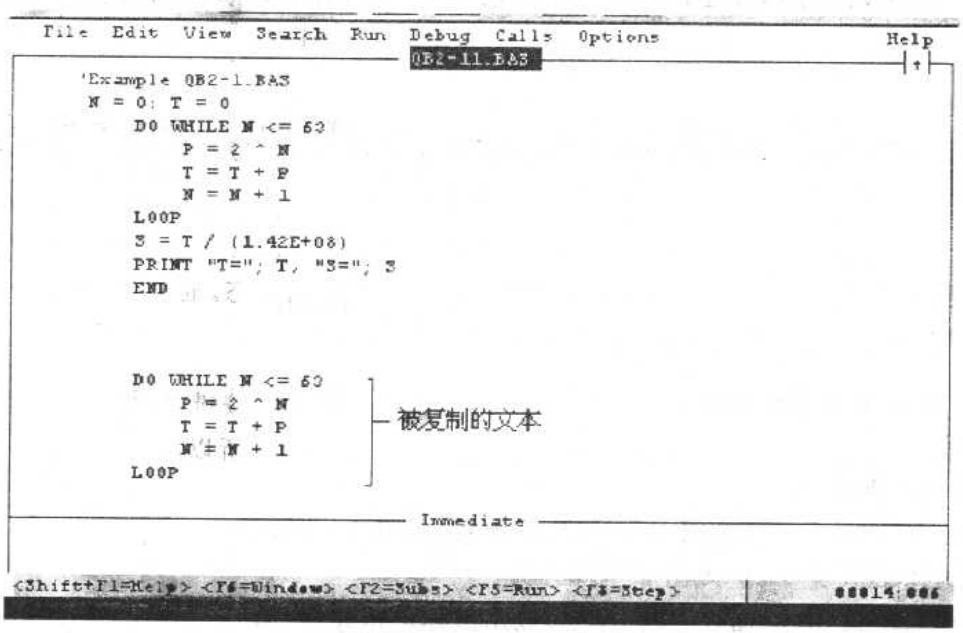


图 2—10　复制后的文本

文本块不仅可以在当前文本内复制，而且可以复制到不同的模块或其它的文件中去。

2.3.4　文本块的删除

块删除的操作步骤为：

1. 定义文本块；

2. 在 Edit 菜单中选 Clear 命令，按回车键即可。

例：把图 2—10 中已复制的内容删除

1. 定义块，如图 2—11。

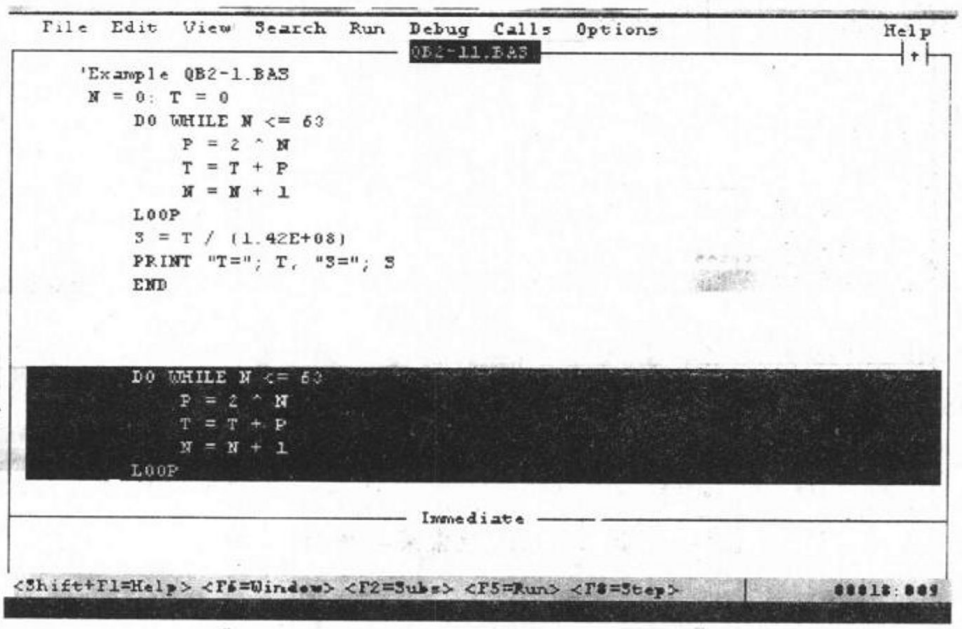


图 2—11　选定删除的文本

2. 在 Edit 菜单中选 Clear 命令，按回车键，文本块就被删除，但是这时该删除的文本块仍然保留在剪贴板中，进入 Edit 菜单，Paste 仍处于激活状态。也就是说，只要剪贴板还没有被新内容替换之前，还可以再次复制，恢复该文本的内容。

定义了文本块，不用 Clear 命令，还可以直接按键盘上的 Del 键删除该文本块，不过这时该文本块并没有保存在剪贴板中，所以不能恢复。

2.3.5　文本的查找与替换

● 文本的查找

【功能】

- 查找一个字符、一个词、一组字或字符串在文本中的位置。
- 可以对查找指定限制条件，例如要不要字母大小写相同、如何分隔词。

· 选择查找范围，可以在当前窗口，或有多个模块的其它模块。

【操作方法】

1. 进入菜单条，选 Search(搜索)菜单，显示菜单画面如图 2—12，该菜单五个命令的名称列在图 2—13 中。

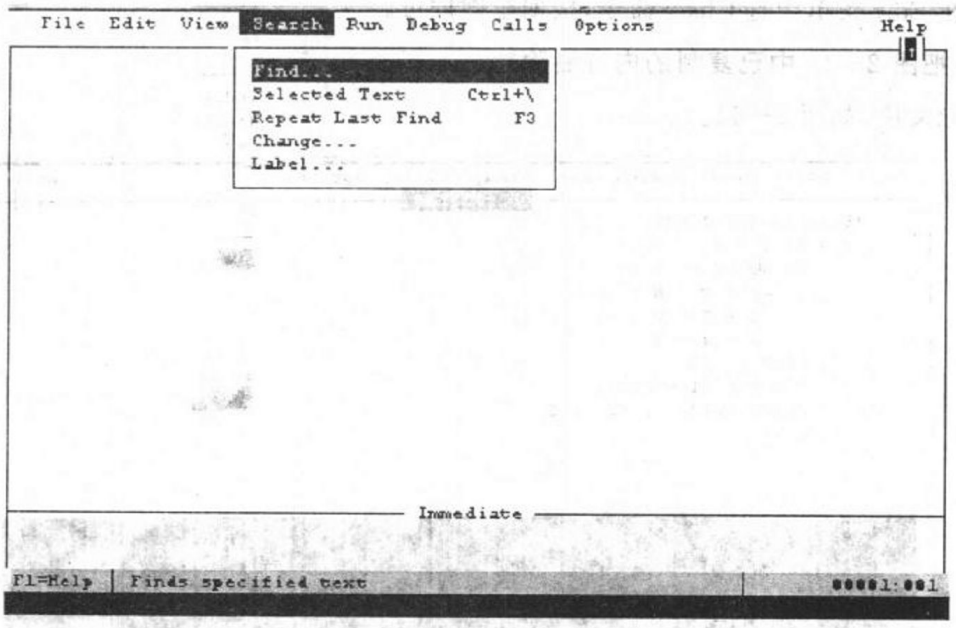


图 2—12　搜索菜单画面

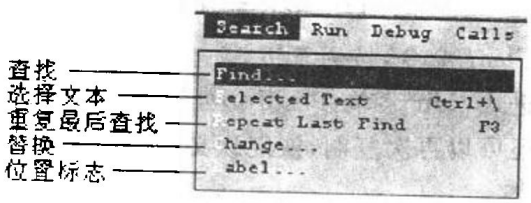


图 2—13　搜索菜单中的命令

2. 选 Find(查找)命令，进入图 2—14 的对话框。

3. 在对话框里填入要查找的内容、条件、范围后，按回车键，如果找到了，该处的字符就呈黑白反视；如果没有找到，将显示找不到的信息：

"Match not found"

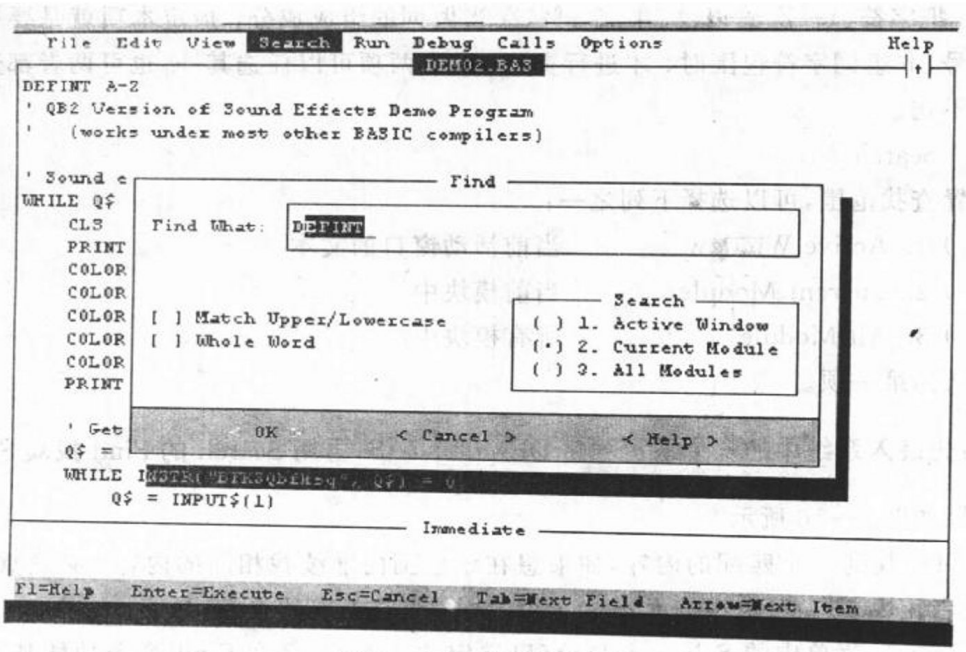


图 2—14　查找对话框

对话框的内容说明如图 2—15。

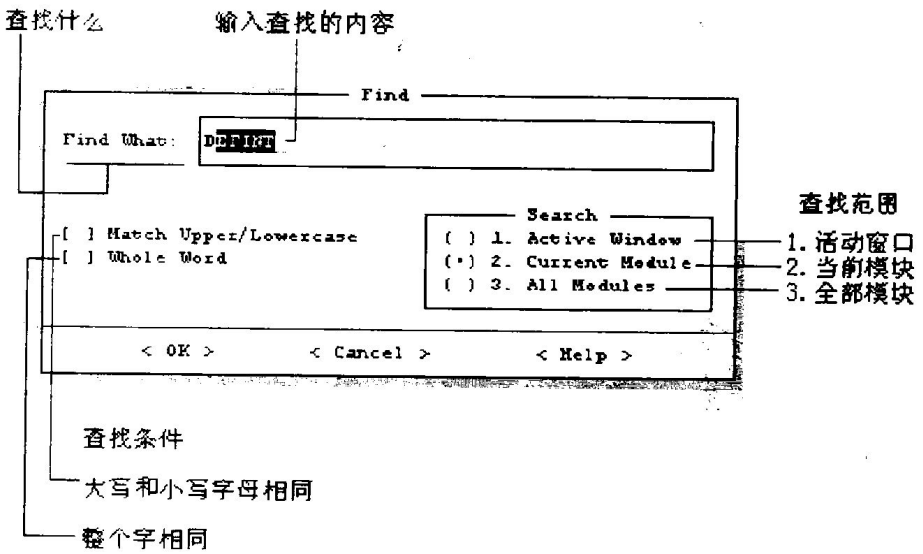


图 2—15　对话框的内容说明

该框有以下三部分内容需要填入：

□　Find What(查找什么内容)

填入要找的字符或词

□　[ ] Match Upper/Lowercase

若选择该项，表示查找的内容必须大小写字母都相同

□　[ ] Whole Word

QB 把字符 A—Z、a—z、!、#、$、%、& 当做词的组成成分，指定本项就是要被空格、标点符号、非组词字符包围时，才进行查找。以上两项可以任选其一，也可两者都选，也可两者都不选。

□ Search

设置查找范围，可以选择下列之一：

( ) 1. Active Window　　当前活动窗口的文本

( ) 2. Current Module　　当前模块中

( ) 3. All Module　　所有模块中

默认为第一项。

例：先装入系统中的一个演示程序 DEMO2.BAS，再用 Search 的 Find 找 Q $

结果如图 2—16 所示。

1. 只能找到一个匹配的内容，如果想在该范围内继续找相同的内容，必须重复上述操作，或者用 Search 菜单中的 Repeat last find(重复上次的查找)命令。

2. Search 菜单中的 Selected Text(已选定的文本)命令和 Find 命令功能基本相同，不同的只是查找前可以先选定查找内容，例如在一行中用 Shift＋→，将所选字符变为黑白反视，所选文本只能在一行内。

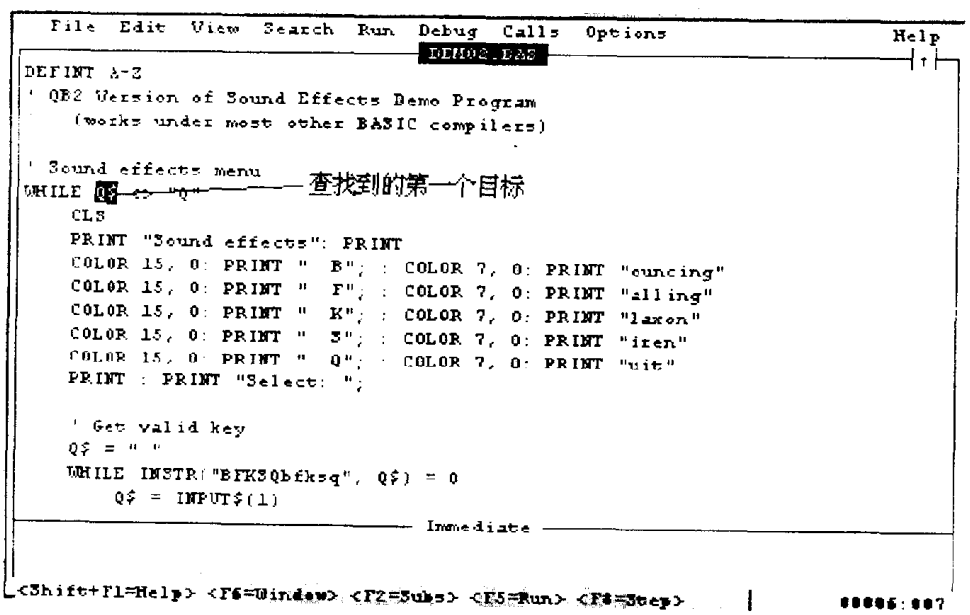


图 2—16　查找到文本的示例

● 文本的替换

【功能】

1. 首先搜索要替换的文本，找到该文本第一次出现的位置。

2. 对找到的文本确定是否用新的文本替换。

3. 可以连续进行查找和替换操作，直到指定的范围搜索完。

【操作方法】

1. 进入 Search 菜单，选 Change(替换)命令，出现图 2—17 的对话框，它的画面与 find 命令基本相同，只是多一个替换内容的输入项。

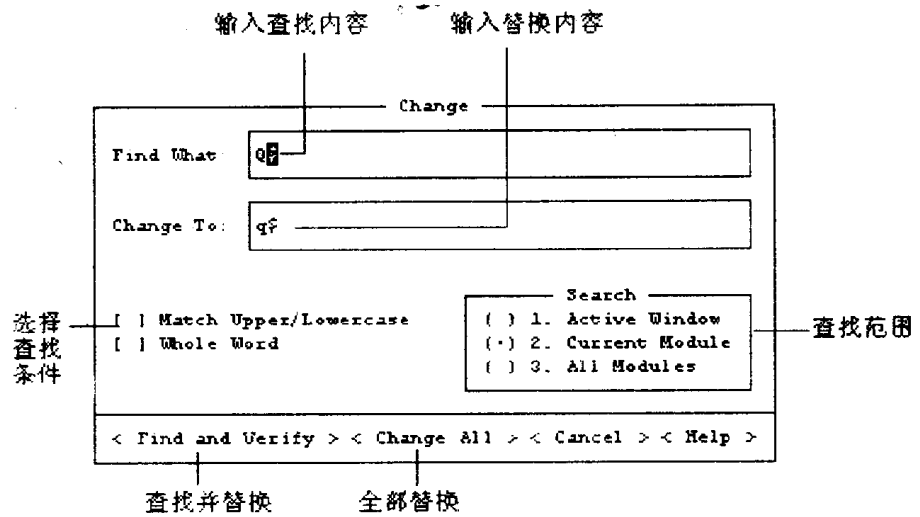


图 2—17　替换的对话框

2. 在该对话框里，填入查找的文本和要去替换的文本，确定条件和范围，按回车键，显示图 2—18 画面，在画面中又弹出一个选择框，由你任选。

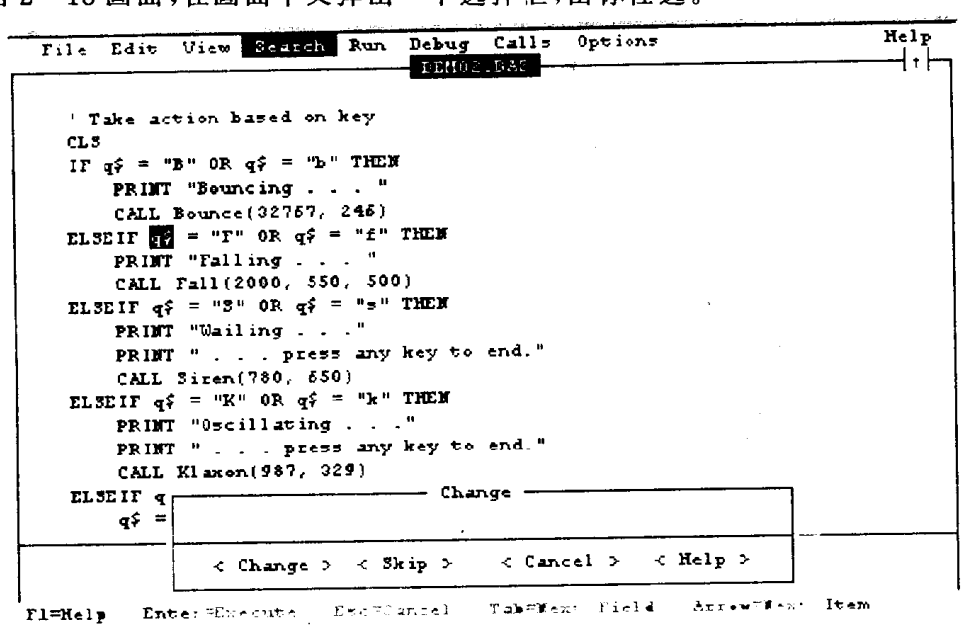


图 2—18　替换时的提示信息

□　Change

对当前文本进行替换操作

□　Skip

跳过当前文本，即对当前文本不进行替换

□ Cancel

放弃替换操作，退出该功能

□ Help

进入联机帮助

3. 当一个文本替换操作完，光标自动移到下一个查找到的位置继续进行替换，直到整个范围查找完，显示一个提示信息"Change complete"，表示操作完毕。

例：将 DEMO2. BAS 程序中的 Q $ 全部用 q $ 替换

结果如图 2—19 所示。

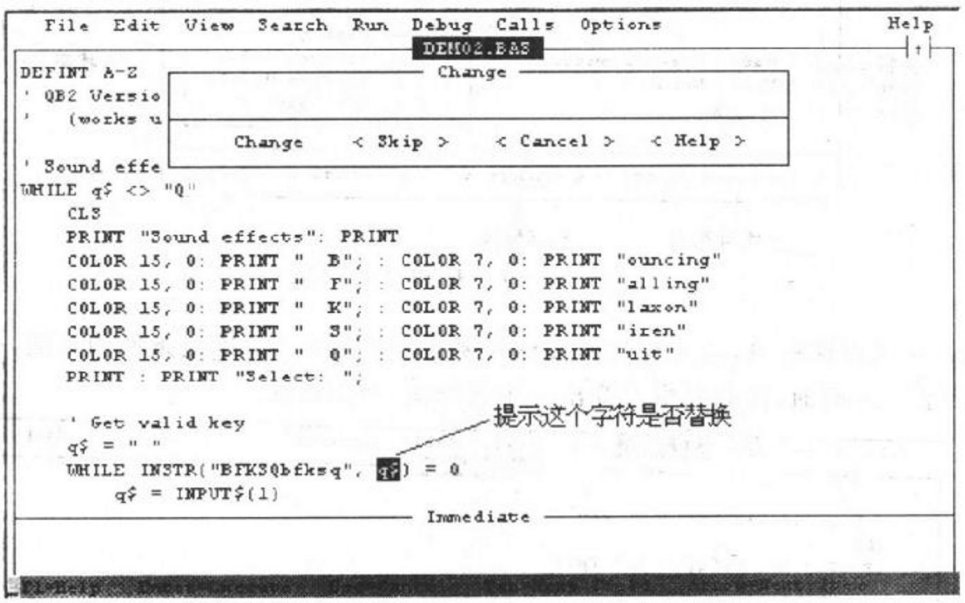


图 2—19　替换示例

第三章

文件的管理

本章内容：

- ▶QB 文件的类别，分类管理的方法
- ▶模块化程序设计的思想和它的优点
- ▶子程序的定义和调用方法
- ▶主模块与其它模块的区别
- ▶模块的建立、保存、组装、取消的方法
- ▶模块、包含文件、文档的特点和区别
- ▶文件格式的特点、用法
- ▶文件的合并、打印的方法

3.1 程序的模块化

儿童时代玩过积木游戏都知道，用一些基本形状的小块，如长方形、正方形、三角形、圆柱形、圆球形等，就可以组合成各式各样的复杂结构，厂房、校门、桥梁、火车等，可以灵活多变，花样翻新。从这里我们得到了启发，在程序设计中，也把一些基本的简单的程序作为模块，用这些模块装配成各式各样的功能强大、结构复杂的程序。现在我们把第一章的几个独立的程序改成为模块，装配成一个能画会唱的多功能程序，并建立一个菜单，可以通过选择菜单，方便地执行每一个模块程序。图 3—1 表示它的结构。

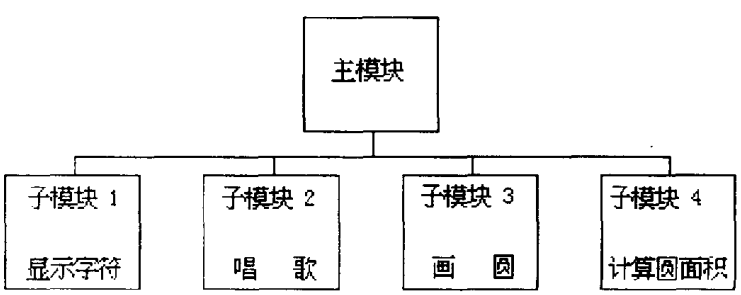


图 3—1 模块演示程序的框图

● 模块演示程序

把第一章中的唱歌、画图、写字、算题四个独立的程序变成一个多模块的程序。

```
'Example QB3—1.bas
DECLARE SUB calarea (area)
DECLARE SUB drcircle ()
DECLARE SUB song ()
DECLARE SUB dispc ()
CLS
DO
LOCATE 6, 25:    PRINT "1.唱  歌"
LOCATE 7, 25:    PRINT "2.画  图"
LOCATE 8, 25:    PRINT "3.写  字"
LOCATE 9, 25:    PRINT "4.算  题"
LOCATE 10,25:    PRINT "5.退  出"
LOCATE 11,20:    PRINT "请输入你的选择(1—5):";

INPUT n
SELECT CASE n
CASE 1
  CALL song
CASE 2
  CLS
  CALL drcircle
CASE 3
  CLS
  CALL dispc
CASE 4
  CLS
  CALL calarea(area)
END SELECT
LOOP UNTIL n = 5
END

SUB calarea (area)
    CONST PI = 3.14159
    LET R = 2
    LET area = PI * R * R
    PRINT "Area="; area
END SUB
```

```
Ⅲ  SUB dispc
       PRINT "Hello! How are you."
       PRINT "Welcome to learn Quick BASIC!"
       PRINT "Wish you successful!"
   END SUB

Ⅳ  SUB drcircle
       SCREEN 1
       COLOR 1, 2
       CIRCLE (100, 100), 50
   END SUB

Ⅴ  SUB song
   PLAY "MNT150L4O3"
   PLAY "cdeccdec"
   PLAY "efg2efg2"
   PLAY "g8a8g8f8ecg8a8g8f8ec"
   PLAY "CO2GO3C2CO2GO3C2"
   END SUB
```

● 程序的说明

演示程序由五个模块组成：

模块Ⅰ：主模块，它的功能是建立菜单，选择调用子模块。

模块Ⅱ：子模块，执行计算圆面积的功能。

模块Ⅲ：子模块，执行显示字符的功能。

模块Ⅳ：子模块，执行画圆的功能。

模块Ⅴ：子模块，执行唱歌的功能。

3.1.1　如何建立模块

● 模块的种类

QB 中的模块可以分为两类

□ SUB 子程序

以 SUB 关键字开头，END SUB 结尾的程序段；

□ FUNCTION 子程序

以 FUNCTION 关键字开头，END FUNCTION 结尾的程序段。

这两类子程序的调用方法不同，所以使用场合也不一样，一般来说，SUB 子程序能返回多个参数的值，FUNCTION 子程序只能返回一个函数的值，因此，SUB 子程序用于需要有多个参数传送的场合，FUNCTION 子程序用于较简单的场合。本章先介绍 SUB 子

程序的使用，FUNCTION 子程序的用法在第七章介绍。

● SUB 子程序的书写格式

```
SUB subname[(a1,a2,…,an)]
        .
        .
        .
  [EXIT SUB]
        .
        .
        .
END SUB
```

【说明】

1. 第一行以关键字 SUB 开头，末行要以 END SUB 结束。

2. 子程序名(subname)，例如演示程序中的四个子模块分别有 calarea、dispc、drcircle、song，给子程序取一名字便于识别不同的子程序，取名的规则和变量名相同，必须以英语字母开头，后跟英语字母或数字，最多可有 40 个字符，但是，建议你不要取得太长，以见名知意，简明易懂为原则。

3. 形参表(a1,a2,…,an)，子程序需要和调用它的其它程序传送参数，在这里称为形式参数，简称为形参，它可以有多个，所以用 a1,…,a2,an 表示，其中…是写格式时的省略符号，表示可以有多个形参，实际编程时，你千万不要写这个省略符号，只需有几个形参就写几个，但是，都要用圆括号括起来，例如模块 Ⅱ 中的 calarea(area)，area 就是一个形参。

在格式中形参表又加了一对方括号[　]，它的含义可以省略，也就是说，SUB 子程序还可以不要形参，这样在子程序名后什么都不写，成为无参子程序，例如演示程序中的模块 Ⅲ、模块 Ⅳ、模块 Ⅴ 都是无参子程序。

4. EXIT SUB 是一个可选项，必要时用它中途退出子程序，电脑不再执行此语句之后的程序，没有必要的话，此项不写。

3.1.2 模块的输入

● 直接输入法

前两章输入的程序都不含子程序，它们的编辑、保存、装入方法大家可能熟悉，但是，当你输入一个带有子程序的程序时，会遇到一个奇怪的现象，例如，正在输入上面的演示程序，如图 3—2 所示，敲完主程序的 END 之后，接着敲 SUB drcircle，刚一按回车键，屏幕一下变了，图 3—2 画面上的程序都不见了，出现图 3—3 画面。

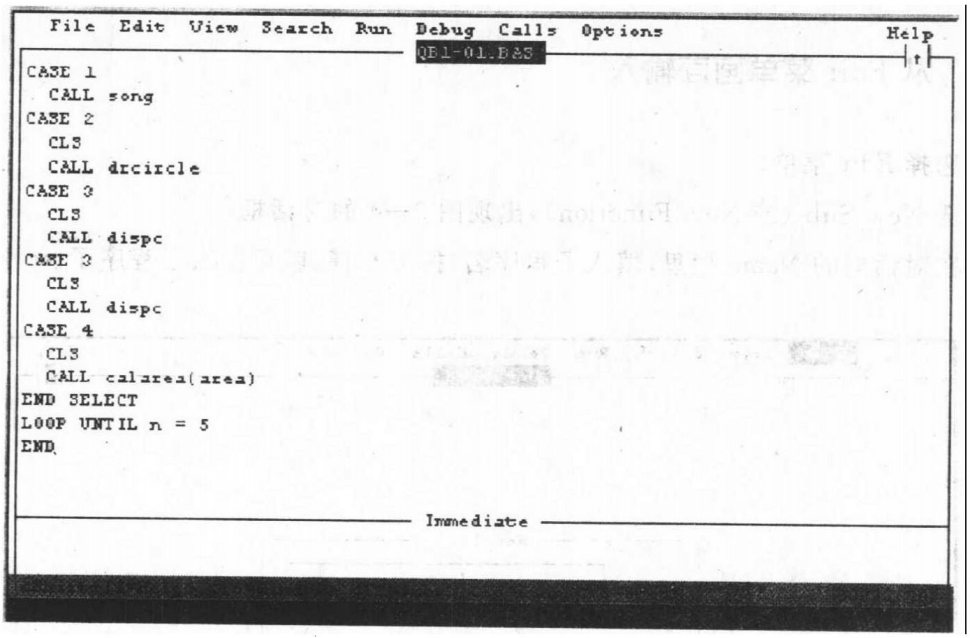


图 3—2　录入主程序时的画面

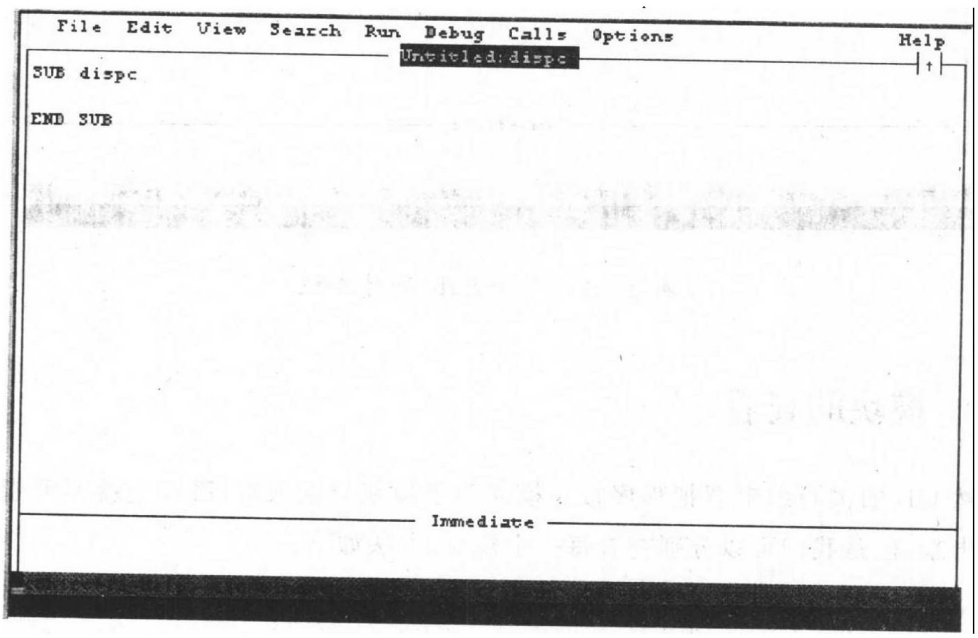


图 3—3　键入 SUB 后的画面

真的把录入的程序丢失了吗？别着急，这是 QB 的灵巧编辑器把每个模块自动分别进行保存，你刚刚录入的主模块和它的程序名已经存放在一个扩展名为.MAK 的同名文件中了。所以，虽然程序在屏幕上消失了，但并没有丢失。

因此，输入子程序，只要键入子程序名，按回车键，出现图 3—3 画面后，在 END SUB 的前行输入文本就可以了。输入完一个子程序，接着用相同方法再输以下的子程序。

● **从 Edit 菜单画面输入**

1. 选择 Edit 菜单；
2. 选 New Sub（或 New Function），出现图 3—4 的对话框；
3. 在对话框的 Name 栏里，填入子程序名，按回车键，就可输入子程序了。

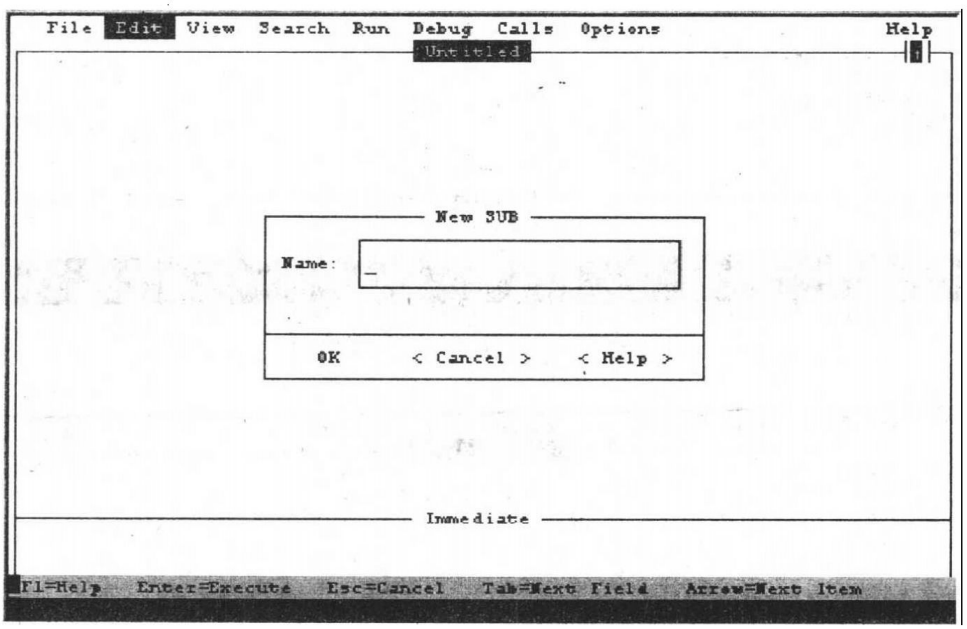


图 3—4　New Sub 的对话框

3.1.3　模块的查看

既然 QB 的灵巧编辑器把程序按主模块和子模块分别保存，画面上就不可能从头到尾显示出来，但是我们可以分别查看每一个模块，方法如下：

1. 选择 View 菜单，显示图 3—5 画面，它有如下两个命令：

   Subs　　列出全部模块的名字；

   Next Sub　　依次显示当前的下一个子程序。

2. 选中 Subs 命令，出现图 3—6 画面，可以看到全部模块名。
3. 用光标选你想查看的模块，按回车键，该模块的程序就显示在观察窗口。

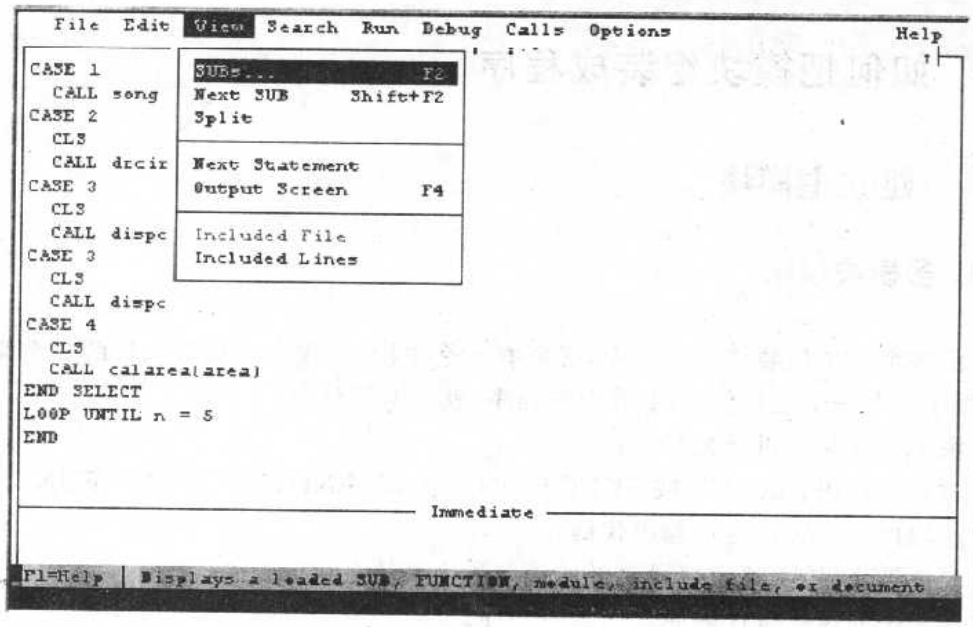


图 3—5　View 菜单

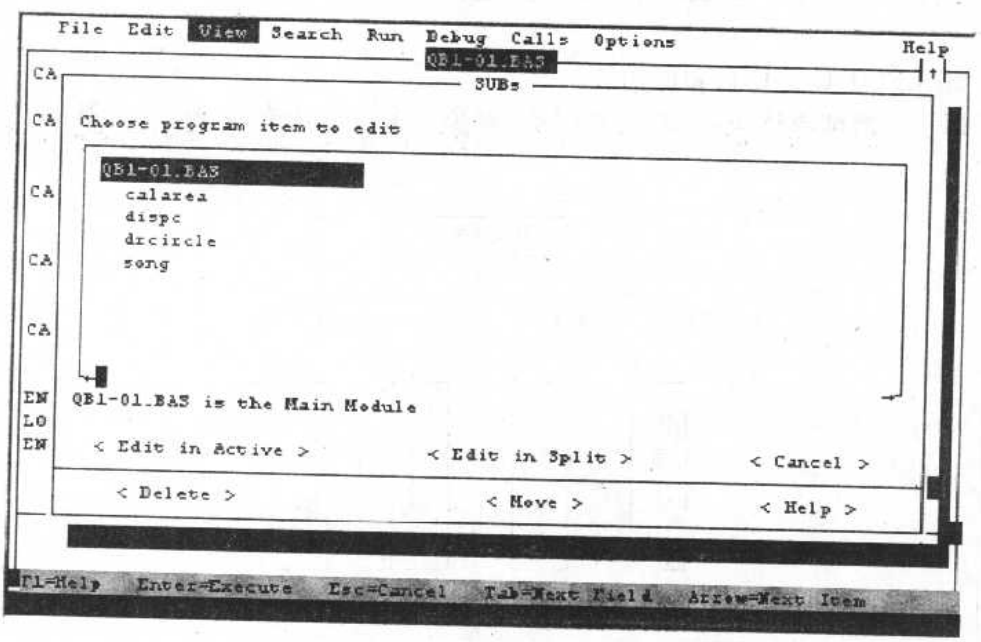


图 3—6　Subs 模块列表

3.2 如何把模块组装成程序

3.2.1 建立主模块

● 多模块程序

在由多个模块组装成的程序中，必须有一个主模块，程序才是可执行的。在简单的情况下，程序只有一个主模块，1.1 节中的程序，就是这种情况。

模块中的语句可以分为两级：

1. 过程级代码：以 SUB 或 FUNCTION 开头，以 END SUB 或 END FUNCTION 结尾的程序段内的语句称为过程级代码；

2. 模块级代码：上述程序段之外的语句称为模块级代码。

主模块中的模块级代码通常称为主程序。

主模块以 END 语句结束。

● 用 SELECT CASE 语句建立主模块

1. SELECT CASE 语句的功能

根据一个测试表达式，产生多个分支，就象一个多路开关，如图 3—7 所示。

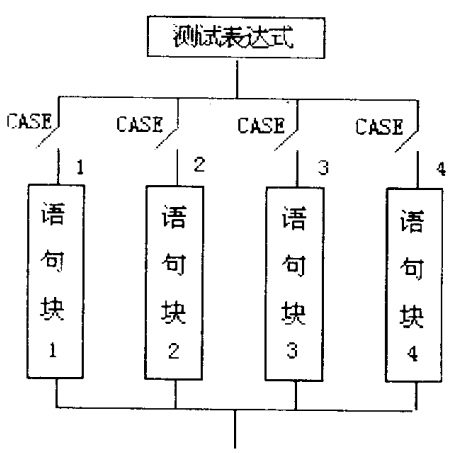


图 3—7　多分支示意图

2. SELECT CASE 语句的格式

```
SELECT CASE <测试表达式>
    CASE        <情况 1>
                •
                •   <语句块 1>
                •
    CASE        <情况 2>
                •
                •   <语句块 2>
                •
    CASE        <情况 3>
                •
                •   <语句块 3>
                •
                ...
    [CASE ELSE
                    <语句块 n>]
END SELECT
```

可以包括多个CASE<情况>及相应的语句块

请你对照前面演示程序的主程序来看看：

```
SELECT CASE n
    CASE 1
      CALL song
    CASE 2
      CLS
      CALL drcircle
    CASE 3
      CLS
      CALL dispc
    CASE 4
      CLS
      CALL calarea(area)
END SELECT
```

由上面对照可以看出：

1.＜测试表达式＞是一个变量 n，根据 n 的值，分别转移到不同情况的语句块去执行；每个 CASE 后的＜情况＞，都是常量，如 1、2、3、4，程序执行时，如果从键盘上选择“1. 唱歌”，键入 1，于是变量 n 的值就为 1。那末，CASE 1 的条件得到满足，程序就执行在它下面的语句块，只有一个语句 call song，这是一个调用子程序 song 的调用语句，电脑就

去执行子程序 song，开始唱歌；如果输入 2，则 n 等于 2，执行 CASE 2 后的语句，调用子程序 drcircle，画一个圆；如果输入 3，则 n 等于 3，执行 CASE 3 后的语句，调用子程序 dispc，显示字符；如果输入 4，则 n 等于 4，执行 CASE 4 后的语句，调用子程序 calarea，计算一个圆的面积。

2. ＜测试表达式＞可以是数值表达式如 2＊n＋1，也可以是字符串表达式如 A＄等等。但不能是关系表达式或逻辑表达式。

3. ＜情况 1＞、＜情况 2＞…可以是以下格式中的任意一种：

＜表达式＞〔,＜表达式＞〕…

＜表达式＞TO＜表达式＞

IS　＜关系运算符＞＜表达式＞

4. 执行完一种情况，就退出该分支；如有两个及两个以上的＜情况＞条件都满足，则只执行最先满足＜情况＞条件的分支；当所有 CASE＜情况＞的条件都不满足时，执行 CASE ELSE 语句后面的语句块，执行完后退出 END SELECT 结构，往下继续执行。CASE ELSE 语句是可选的，如果没有必要，可以略去不写。

5. 有一个 SELECT CASE 语句，必须有一个 END SELECT 语句与之配对。

例：某幼儿园儿童按年龄编班，2～3 岁的为小班，4 岁的为中班，5～6 岁的为大班，输入某儿童的年龄，计算机给出应编进哪个班级。

【算法】

把人园儿童的年龄设为 OLD，以该变量为测试表达式，按 2～3，4，5～6 三种情况来应用 SELECT CASE 语句写程序。

【源程序】

```
'Example QB3－2. bas
INPUT "How old are you?";OLD
SELECT CASE OLD
CASE 2,3
  PRINT "Enter lower class"
CASE 4
  PRINT "Enter Middle class"
CASE 5 TO 6
  PRINT "Enter Higher Class"
CASE ELSE
  PRINT "Can not enter!"
END SELECT
END
```

【语句说明】

INPUT 语句的功能是从键盘上输入数据，程序执行到该语句时，屏幕就会回到 DOS

状态，并出现提示信息，等待你输入程序需要的数据，见图 3—8。

```
C:\CL\QB45\qb
How old are you?_←——等待输入一个年龄的数值
```

图 3—8　程序 QB3—2.bas 的执行过程一

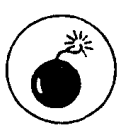
这时你就可以在光标位置输入数据，键入一个数 5，按回车键，程序继续执行，你就能看到它的结果。

```
C:\CL\QB45\qb
How old are you? 5,5  ←——输入了两个数，出错

Redo from start  ←——从头再输入
How ole are you? 5


Press any key to continue
```

图 3—9　程序 QB3—2.bas 的执行过程二



键盘输入的数据类型要与 INPUT 语句中变量的类型相同，数据个数也要相等，不能少于或多于变量的个数，否则，电脑不买你的账，出现一个提示信息“Redo from start”(从头再输入数据)，如图 3—9。

此时，你要检查一下数据类型是否正确，数据个数是否相等，然后按 INPUT 语句中变量的类型和个数，正确地重新输入，如果你又没有输对，它又要重复出现该信息。如果你还没懂得起，它就一直这样提示你，千万别着急，乱按一通键也无济于事，到你走头无路时，只好来个“刹车”，按 Ctrl+Break 键，就可以退出来。

3.2.2　如何调用子程序

一个单独的子程序是不能在电脑中执行的，它可以被其它的程序使用，我们要使用一个子程序，就称为调用子程序，不仅主程序可以调用子程序，而且子程序也可以调用子程序，子程序还可以调用它自身，这种情况，我们称为递归调用。

● 调用子程序的方法

1. 用 CALL 语句调用 SUB 子程序。

CALL 语句格式：

CALL Subname〔(实参表)〕

例如　CALL Calarea(area)

其中:CALL 是语句的关键字。

Subname 是被调用的 SUB 子程序名。例如 Calarea 就是程序名。

( )内是实参,比如上面括号里的 area,它和 SUB 子程序中的形参个数要相等,位置要一一对应,类型也要相同。

如果子程序是无参子程序,那么,实参也就省略了。

CALL 语句的功能是把流程控制转移到 SUB 子程序中去,把实参和形参相结合。

2. 省略关键字 CALL,直接写 SUB 子程序名。这时实参表的圆括号也必须省略。

格式如下:

SUBNAME [<实参表>]

例如　Calarea area

例:计算自然数 N 的阶乘 N!

【源程序】

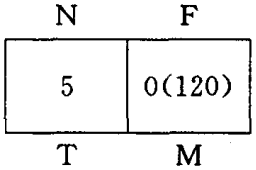
```
'Example QB3—3. bas
DECLEAR SUB Fact(T,M)
INPUT "Enter a number:",N
Fact N,F                    '调用子程序 Fact,N,F 是实参
PRINT "The Factor is";F
END

SUB Fact(T,M)
M=1
FOR I=1 TO T
  M=M*I
NEXT I
END SUB
```

【程序说明】

1. 主程序中的实参 N,F 和子程序中的形参 T,M 要按位置一一对应进行结合,结合的过程如下:

主程序的实参存放的单元



子程序的形参存放的单元　　T　　M

它们是以共用相同的存储单元来实现结合的,数据可以双向传送,比如,当实参 N=5(用 INPUT 语句从键盘输入)、F=0(在主程序里没有赋值,初始化为 0),执行子程序时,形参 T 就取得值为 5,子程序执行结果 M 的值为 120,因此,退出子程序,实参 F 也有了和 M 相同的值,这样主程序的输出语句 PRINT 就能得到 5 的阶乘值为 120。

2. FOR——NEXT 是循环语句,它的用法见第七章。

3.3 QB 中源文件的类别

QB 中的源文件根据管理的方式分为三类：

·模块(Module)

·包含文件(Include)

·文档(Document)

根据存放的格式分为两类：

·快速格式

是一种非 ASCII 码的 QB 专用格式，只供 QB 进行编辑，不能用其它文本编辑程序(如 WS、WPS 等)进行修改，这种格式的特点是存取速度快，但是包含文件和文档文件不能用这种格式存取。

·文本格式

是一种 ASCII 码的文本文件，可以用其它文本编辑程序(如 WS、WPS 等)进行修改，包含文件和文档文件只能用这种格式存取。

模块可以用这两种格式的任何一种存取。

3.3.1 模块

● 模块和程序的关系

从 QB 文件管理的角度看，模块是一种存取单位，是程序的一个部件，它把每一个文件都认为是模块。比如一个单一的主程序，就是一个主模块，一个单一的子程序，也可以作为独立的文件，构成一个模块，由几个模块构成的程序，也可看成一个模块，只不过是组合式的模块而已。

从 QB 运行功能的角度看，程序是语句的集合，用 QB 语句构成的程序，称为源程序，存放源程序的文件，称为源文件(常常以.BAS 为扩展名的文件)。一个程序可以由多个模块组成，但是，必须指定一个主模块，程序才是可执行的。程序中的模块可以有两种保存方式：

·所有模块和主程序保存在一个文件中，每次装入时，所有模块都一起被装入内存，可以用 View 菜单下的 Subs 命令，看到模块的名字，也可以直接执行。上一节的内容已详细介绍过了。

·每一个模块各自保存为不同的文件，每次装入时，要用 File 菜单中的 Load File 命令把每个模块组装在一起，并指定主模块后才能执行，组装后也可以在 View 菜单下的 Subs 命令，看到模块的名字，下面立即会介绍它的操作方法。

包含文件不是模块。

● 文件的保存

在前面已经介绍了用 Save As 命令保存文件的方法，现在介绍完备菜单中的另外两个保存文件的命令：Save、Save all，以及它们用法上的区别。

它们的功能和区别如下：

1. Save

用标题栏里的文件名，把当前观察窗口的内容写到一个磁盘文件中去。它和 Save As 不同的是：

如果你是装入的某个文件，选择 Save，一按回车键，不出现对话框，文件就保存了，文件名和装入时相同，保存修改后的内容；如果你是新建的文件，标题栏是 Untitled，选择它，一按回车键，出现对话框，和 Save As 的对话框相同。

选择 Save As，一按回车键，将弹出对话框，你可以更改文件名和选择保存格式。如果你是装入的某个文件，又经过了修改，保存时可以另取文件名，这样，保存了修改后的内容，修改前的原文件的内容也能保存在原文件名的磁盘文件中。

2. Save All

用于保存多模块文件，这些多模块文件是当前装入的、模块名为 View 菜单的 Subs 中的文件，它和 Save 命令相同的是，如果是一个新文件，则弹出对话框，如果是装入的已有文件，就不出现对话框，它把所有作过变动的文件都进行保存。

● 用 Load File 命令组装程序

1. 我们设定如下一个多模块程序：

```
'Example QB3—4.BAS
DECLARE SUB volume (s())
DECLARE SUB Density (s(), w)
'main program
DIM s(3)
c = 52
s(1) = 3: s(2) = 3: s(3) = 6
volume s()
Density s(), c
END
```

```
SUB Density (s(), w)
vol = s(1) * s(2) * s(3)
Dens = w / vol
PRINT "The Densiyt is"; Dens
END SUB

SUB volume (s())
vol = s(1) * s(2) * s(3)
PRINT "The Volume is"; vol
END SUB
```

2. 用一个文件保存这个程序：

(1) 首先，输入主程序，直到 END；

(2) 选 Edit 菜单中的 New Sub 命令，输入 volume 和 Density 两个子程序；

(3) 在 Save As 画面用文件名为 QB3－4.BAS 存盘；

(4) 用 View 菜单的 Subs 命令，查看模块的目录，我们就会看到如图 3－10 所示的画面。

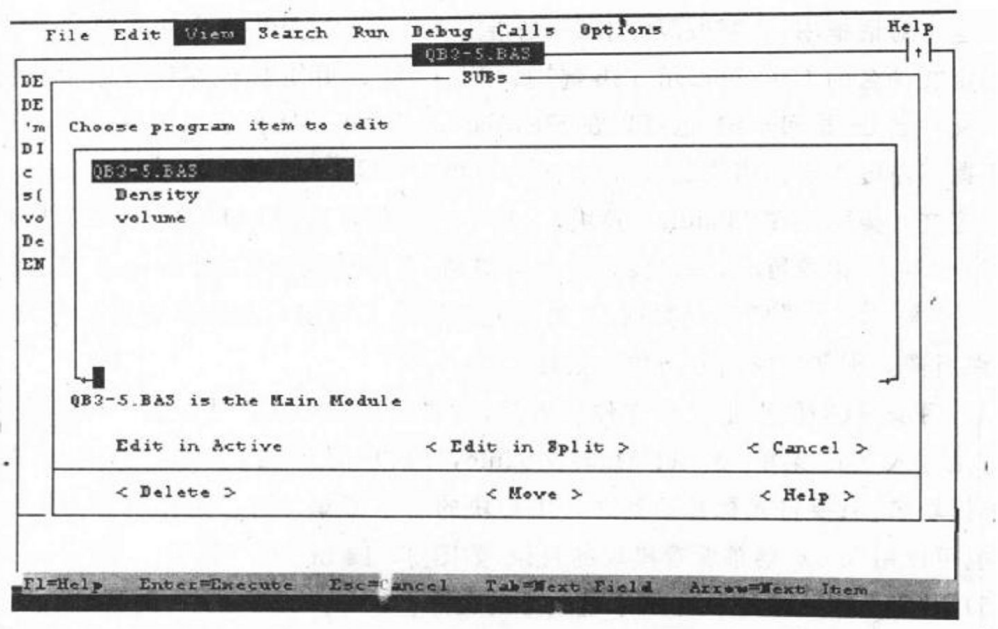


图 3－10　Subs 命令下模块的目录

3. 用三个文件分别保存这个程序的主程序和两个子程序：

(1) 输入主程序，直到 END，用 Save As 存盘，取名为 PQ01.BAS；

(2) 输入子程序，也用 Save As 存盘，取名为 PQ02. BAS 和 PQ03. BAS；

(3) 用 File 菜单的 Load File 命令将三个独立的模块组装成一个程序，当选了 Load File 命令后，出现如图 3—11 的对话框。

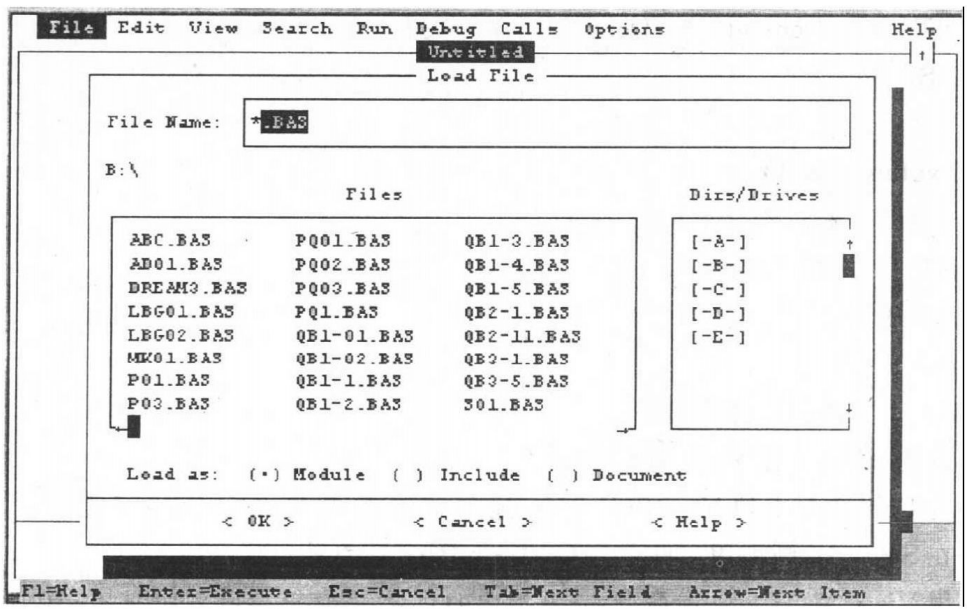


图 3—11　Load File 对话框

在这个对话框中，你要做两件事：一是指定文件名；二是指定文件类别。

指定文件名的方法，可以用 Tab 键进入 files 栏，再用光标键移动黑色亮条到你需要装入的文件名处，按回车键；也可以在 File Name 栏输入你需要装入的文件名，按回车键；或者用鼠标左键单击你需要装入的文件名，再单击<OK>选项。

指定文件类别是在 Module(模块) 、Include(包含文件) 、Document(文档) 三类中任选其一，但是，你要留心，一定要正确选择类别，它的默认选择是 Module，该项前的圆括号中有一“ * ”号，如果要改变类别，只须把光标移动到所要的类别名前的括号内，回车即可，或在所需类别名前的括号内单击鼠标。

(4) 用步骤(3)依次把三个文件装入后，还要在这三个文件中设置一个作主模块，设置方法是进入 Run 菜单，选 Set Main Module，将出现图 3—12 所示的对话框，框内显示出全部模块名，只要将光标移动到选为主模块的名字回车即可。这样就可以执行该程序了，你也可以用 View 菜单看看模块的目录，如图 3—13 所示。

(5) 用 Save All 命令保存这个组装好的程序。

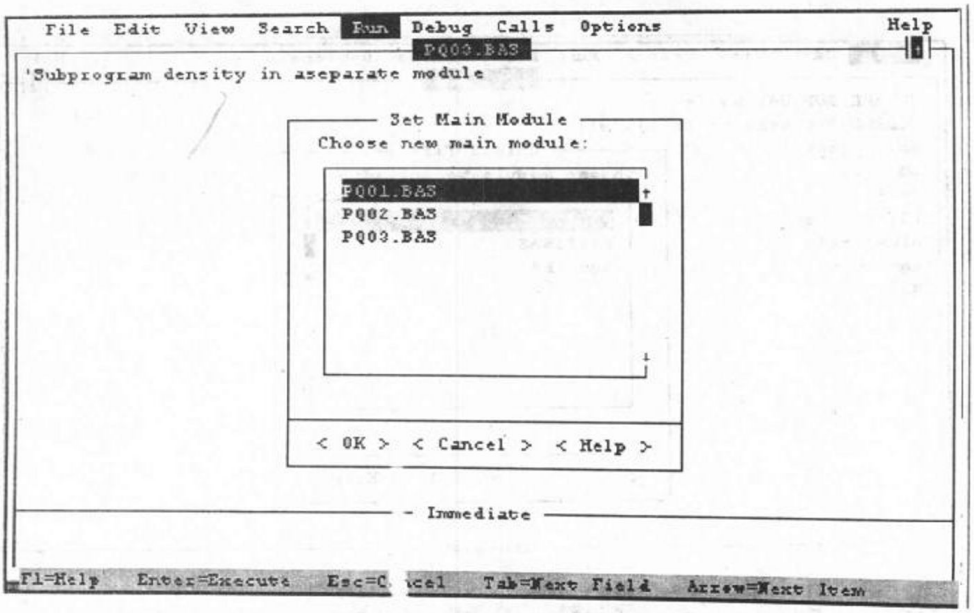


图 3—12　设置主模块的对话框

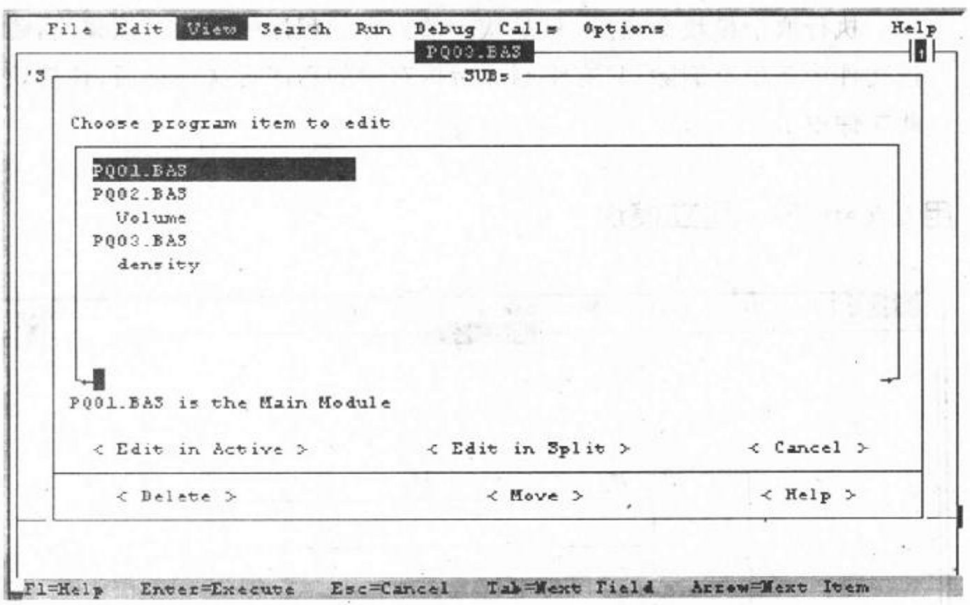


图 3—13　组装后的模块目录

● 用 Unload File 取消模块

对于多模块程序，有时需要作修改，用 Load File 命令可以增加模块，用 Unload File 可以取消模块，当选择了 File 菜单里的 Unload File 后，出现图 3—14 所示的对话框，只要将光标移动到你要取消的模块名上，按回车键即可。

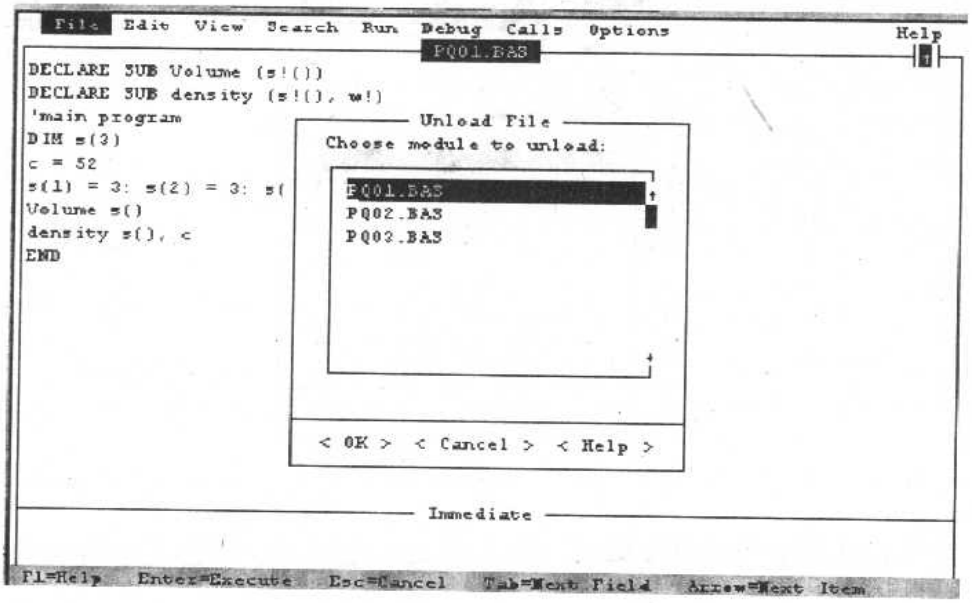


图 3—14　Unload File 对话框

执行取消模块命令后，只是从当前内存的程序中删除了该模块，磁盘上在该文件中还没有删除；只有将删除后内存中的程序再次保存后，该模块就不在此文件中了。

● 用 Create File 建立模块

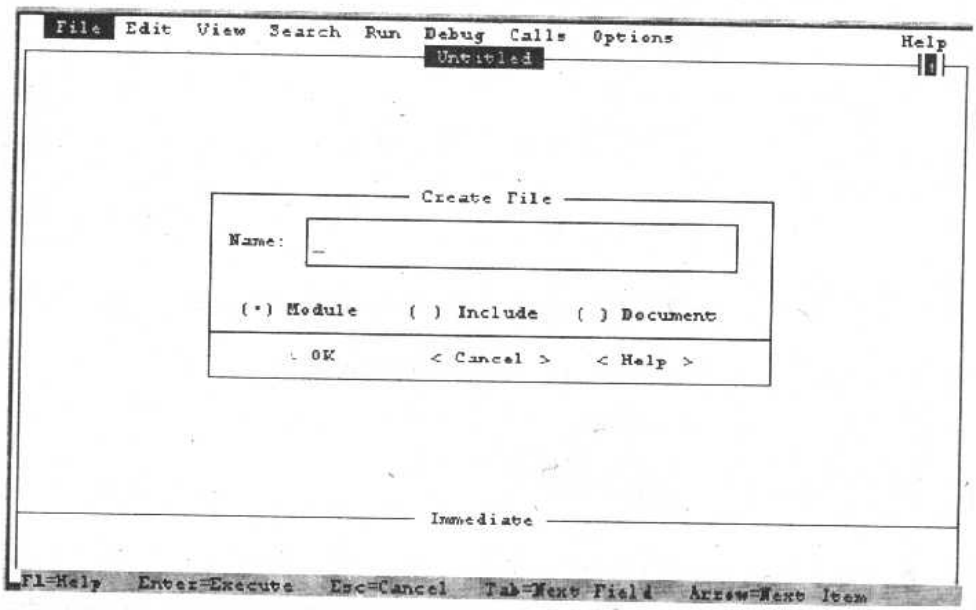
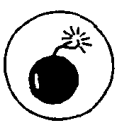


图 3—15　Create File 对话框

File 菜单中的 Create File 用于建立不同类别的文件，所以它可以用来建立模块，也可以用于建立包含文件或文档。以前已经学过的用 Open Program 命令输入程序，只能建立模块，不能建立包含文件或文档。

选择 Create File 命令后的对话框如图 3—15 所示。

在这个对话框中，你要做两件事：一是指定文件名；二是指定文件类别，这个和 Load File 命令对话框的操作相同，已经在前面详细介绍过了，特别要注意的是正确选择文件类别。



当你完成文件名和类别的指定后，出现输入画面，此时文件名就在标题栏里。你就可以开始输入程序，输入完毕，要用 Save 命令保存文件，此文件才能存放到磁盘上，千万不要以为该命令已经把你输入的程序文件都保存在磁盘上了。

3.3.2 包含文件(Include)

● 什么是包含文件

打包这个概念在平常生活中是很好理解的，在程序设计时，也应用了这个概念，我们把一些不同程序或模块中都要用到的语句放在一起，打包成一个文件，就成了一个包含文件，它和模块类似，但它又不是模块，它不能包含 SUB ENDSUB 和 FUNCTION END-FUNCTION 语句，只能用文本格式存取，不能用快速格式存取。因此，给这类文件用一个特别标志，$INCLUDE，电脑编译时一看就知道。我们写程序时，在不同的程序里只要写出这个标志和文件名，不用写相同的语句，这样既减少了程序员的输入工作量，又节省了磁盘空间。QB 提供这种处理附加源文件的功能，对程序中多个模块都共用的公共数据块、变量类型定义语句等，非常方便。

在 QB 中把 $INCLUDE 称为元命令，所以包含文件就是用一条 $INCLUDE 元命令编入程序中的文本文件。

● 建立和保存包含文件

从 File 菜单选择 Create File 命令，并通过对话框(见图 3—15)，输入文件名和确定文件类别，一定要选择 Include，否则，QB 将该文件作为模块处理，在多模块程序存盘时会把包含文件的文件名也装入.MAK 文件，这样的结果只能浪费内存空间，而毫无用处。

选择对话框后，就可以输入程序语句，输入完毕要存盘，只要你在建立文件时就指明 INCLUDE 的类别，那么无论用 Save 或 Save as 命令，它都以 ASCII 格式存盘。

● 观察和编辑 INCLUDE 文件

用 View 菜单中 Included File 命令查看 Include 文件的内容，当此命令在开启时，观察窗的 $Include 元命令行下面将以高亮度显示出包含文件的内容。但不可直接修改，若一定要修改，一个对话框会询问并经同意后将该内容调入观察窗，然后才可修改、编辑。

必须当光标处在 $Include 命令行时 Included File 命令才有效，此时该命令的第一个字母"I"是用高亮度显示的。

3.3.3 文档

● 什么是文档

在 QB 编程环境中，可以保存用 ASCII 码形式的文本，这种文本称为文档，它能在其它的文字处理软件中进行编辑，例如广泛使用的 WordStar、WPS 等，所以 QB 的编辑功能不仅是编辑 QB 的源程序，也可作为字处理软件使用。如果要输入汉字，必须首先启动汉字操作系统，如 UCDOS，然后启动 QB，在 QB 画面里，进入 File 菜单，用 Create File 命令建立档案文件。

● 文档的建立和保存

选 File 菜单中的 Create File 命令，在图 3－15 的对话框里，只要指定文件类别为 Document 即可。

QB 对文档的输入既不作格式化，也不作语法检查。

保存文档时，可以用 Save 或 Save AS 命令，但是，不能以快速格式保存，只能以文本格式保存。

● 注意事项

1. 凡是用文档建立或装入的文件都是不能执行的。

2. 用 QB 快速格式保存的文件，不能用文档装入，如果你在 Load File 命令中用 Document 去装入，就会出现图 3－16 所示的出错信息：

"Bad file mode"，意思是文件类别不对。

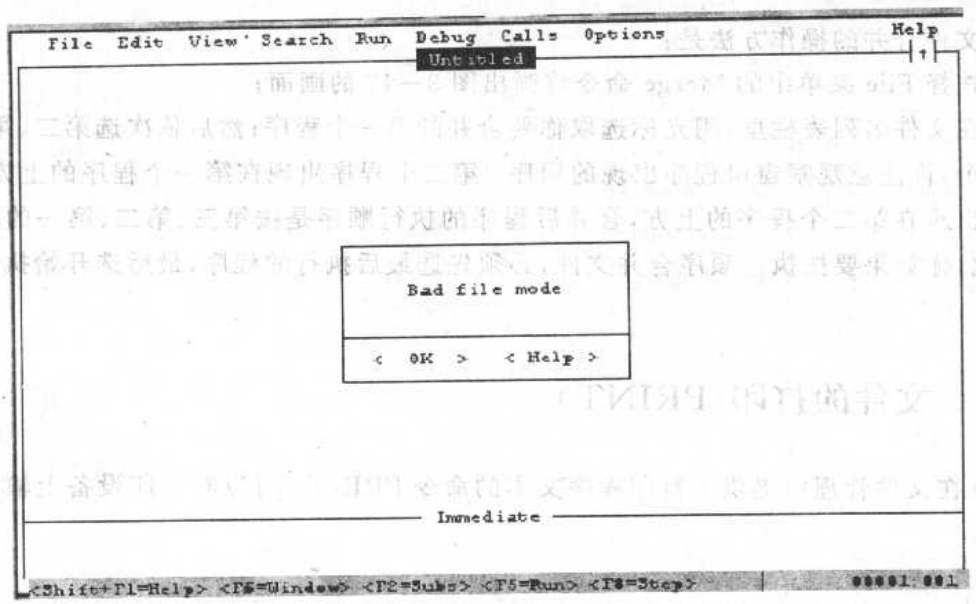


图 3—16　文件类别不对的出错信息

3.4　文件的合并与打印

3.4.1　文件的合并(Merge)

QB 提供把两个或多个以文本格式存放的文件合并为一个程序的功能，但是以 QB 快速格式存放的文件不能用 Merge 合并，如果你企图合并这类文件，将出现出错信息。

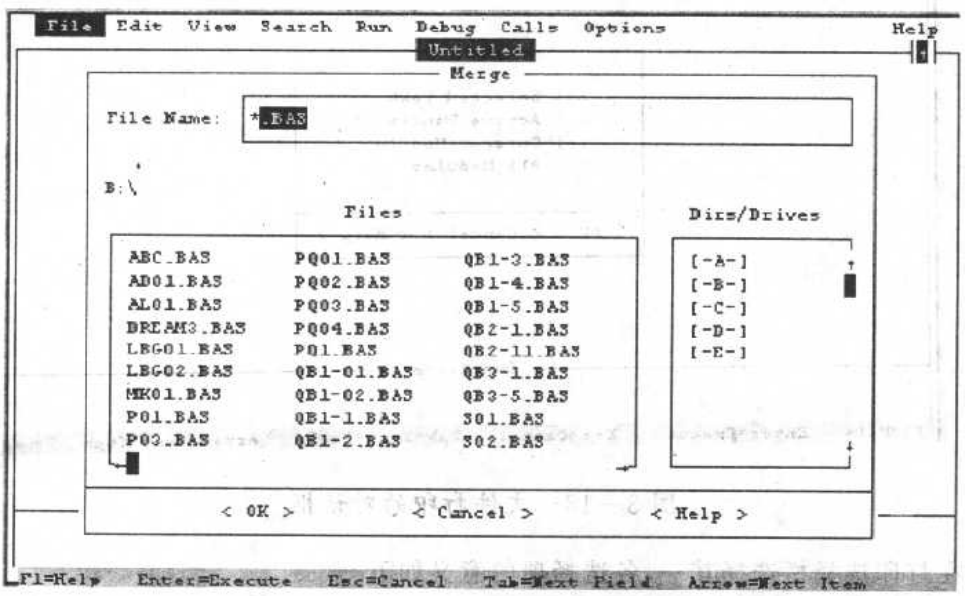


图 3—17　文件合并的对话框

将文件合并的操作方法是：

1. 选择 File 菜单中的 Merge 命令将弹出图 3—17 的画面；

2. 在文件名列表栏里，用光标选取你要合并的第一个程序；然后依次选第二、第三个，每选一个，你注意观察窗口程序出现的顺序，第二个程序出现在第一个程序的上方，第三个程序出现在第二个程序的上方，合并后程序的执行顺序是按第三、第二、第一的排列顺序，因此，你如果要按执行顺序合并文件，必须先选最后执行的程序，最后选开始执行的程序。

3.4.2 文件的打印(PRINT)

QB 在文件管理中提供了打印程序文本的命令 PRINT，可以在打印设备上输出源程序清单。

它和程序中 PRINT 语句的功能不同，它不能输出程序的执行结果。

文件打印的操作方法是：

1. 首先，装入要打印的文件；

2. 选择 File 菜单中的 Print 命令将弹出如图 3—18 的画面；

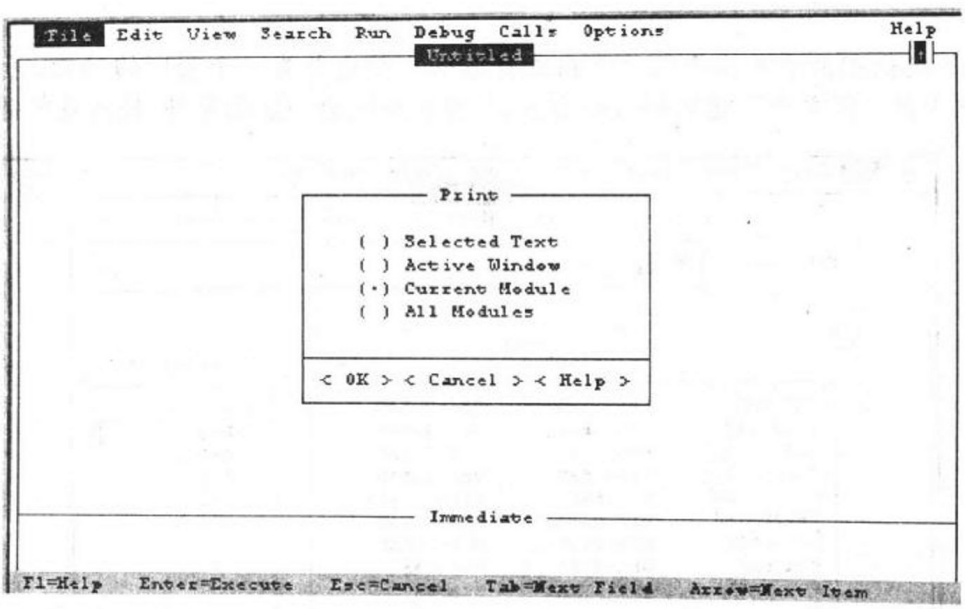


图 3—18　文件打印的对话框

3. 在打印选择栏选择其一，各选择项的意义如下：

□ ( ) Selected Text

打印用块定义选择过的文本

□ ( ) Active Window

打印在当前活动窗口中的文本

□　(　) Current Module

打印与当前活动窗口中的文本和它组合在一起的全部模块

□　(　) All Module

打印在内存中的所有模块、包含文件、文档

第四章

程序的执行与调试

本章内容：

- ▶电脑是怎样执行程序的
- ▶如何控制程序的流程
- ▶查错的方法和如何阅读错误信息
- ▶设置监视表达式、监视点、断点
- ▶单步跟踪和自动跟踪程序执行的方法
- ▶综合利用 QB 的 DEBUG 功能检查和排除程序中的错误

4.1 QB 源程序是怎样被执行的

4.1.1 高级语言与机器语言

● 机器语言

人和人之间是通过语音(口头语言)、文字(书面语言)来互相交流思想、传递信息的，人和电脑打交道，也是用一种特定的语言来互通信息(或称人机通信)。电脑就是用一种能产生、处理、传送、存储数字信号的微电子器件构成的机器。人们用 0、1 来表示数字信号的两种不同状态，然后，用 0、1 来编码，比如，表示数值的二进制数，表示字符的 ASCII 码，表示电脑基本操作的指令码等等，这些编码都是人们的一种约定。我们要电脑做各种事情，实际上，它都在执行规定好的一套指令，这些指令就是由 0、1 组成的一串编码符，象 10111101 之类的指令形式，我们把它称为机器语言。

直接用“1”、“0”组成的机器指令编写的程序，就是机器语言源程序。对电脑来说，这是它唯一能直接“听”得懂的语言。所以，常常称之为面向机器的语言，但是，对使用电脑的人来说，这是十分难懂的语言，它难读、难记、难写，容易出错，不同机型又不通用。显然人和机器之间的通信存在巨大的鸿沟。只有填补了这个鸿沟，使用的人才会方便容易，机器又能懂得，电脑才能发挥更大的作用。为此，人们设计了汇编语言。

把用二进制数表示的指令，用一些符号来表示，如用表示操作的英文缩写来代替指令代码，用 16 进制数表示数字，这种用符号代替后的指令，就叫汇编语言，又称符号语言，用

汇编语言编写的程序，称为汇编语言源程序，常简称为汇编语言程序。这种语言，相对机器语言就容易读、容易记、容易写了。但是，机器却一点也不懂了。因此，电脑是无法直接执行的。一个不懂汉语的外国人到中国来要和中国人直接交谈，那是无法进行对话的，所以，只好求助翻译。在电脑中，也同样采取这种方法，人们编写程序用汇编语言，然后请一位翻译，把汇编语言程序翻译成机器能懂得的机器语言程序，这个翻译过程，叫做“汇编”。汇编后产生的机器代码称为目标程序。翻译可以由人手工完成，但做起来既繁琐单调，又容易出错，因而就研制出了担任翻译的程序，取名叫汇编程序。汇编过程如图 4—1 所示。

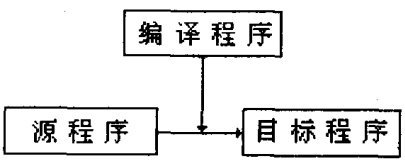


图 4—1　源程序的翻译过程

汇编语言使程序设计工作前进了一大步，但是仍然存在很多缺点，第一、不便于我们了解问题过程的描述，如一个数学公式，汇编语言的表达形式与人们的习惯表达形式差别很大；第二、它仍是面向机器的语言，不同机型，汇编语言也不一样，因而用它编制的程序，没有通用性。为了克服这些不足之处，人们进一步研制出了高级语言。

● 高级语言

它是更接近人的自然语言和数学表达式的一种语言，完全不依赖机器的指令系统。这样的高级语言为人们编制程序提供了很大的方便，编制出来的程序易读易记，也便于修改、调试，大大提高了编制程序的效率，也大大提高了程序的通用性，便于推广交流。从而极大地推动了电脑的普及与应用。

我们看到高级语言离人的理解愈加接近了，但离电脑的理解就越远了。电脑是不能直接理解那些英语单词、数学表达式的。所以，为了填补人机之间的鸿沟，还是得求助翻译。把高级语言翻译成机器能懂的机器语言，这一翻译过程，我们称为“编译”，用高级语言编写的程序称为高级语言源程序(Source)，经过编译后的机器语言程序称为目标程序(Object)。翻译这一工作是由编译程序(Compiler)来做，这个编译程序也是人们设计并事先装入电脑的。没有这一程序，高级语言源程序就无法在电脑中执行。

目标程序(Object)还不能执行，需要把一些标准模块和它连接，形成为可执行程序(Executive)，这一工作是由连接程序(Linker)来做的。

因此，高级语言源程序输入以后，都要经过编译，连接，执行的过程。不过，有的高级语言与此稍有差别，如较早的 BASIC 语言，源程序是翻译一个语句就执行一个语句，边翻译边执行，并不生成目标程序，这一过程叫解释，担任翻译工作的称为解释程序。

QB 既保留了早先 BASIC 的解释功能，又具有编译的功能。下面我们将详细讨论 QB 的这一 特点。

4.1.2 QB 的解释功能和编译功能

● QB 的解释功能

我们在第一章中已经介绍了 QB 程序的输入与执行，当你输入一个语句，按了回车键或将光标移动出该行后，编辑器就开始工作了，首先进行语法检查，如果语法有错，立即显示一个错误信息的会话框；如果没有语法错误，它将把该行翻译成可执行代码，整个程序输入完毕，也就全部翻译完毕。这一切工作都是在内存储器中进行，不需要磁盘，翻译后的程序，也不保存到磁盘上，但它已经是可以马上执行的，只要你发出执行(RUN)的命令，或选择 RUN 菜单中的 Start 命令，或者按 Shift+F5 键。

这就是 QB 的解释执行，它的好处在于很方便地调试程序，不但输入时就可以发现和修改程序，而且程序执行如果出错，QB 也将显示一个错误信息的对话框，你可以立即修改，并再次执行，直到完全正确为止。这给初学者带来了极大的方便，但是，这种解释执行方法，影响执行速度，每执行一次，都要用它解释，不便于软件开发。

● QB 的编译功能

其它的高级语言，如 Pascal、FORTRAN、C 等等，一般都要经过文本编辑、程序编译、连接、执行这样的过程，每一步都要用相应的软件，例如用 Edit、QE、PE、WPS 等输入源程序，再用各自的编译程序(Compiler)进行编译，用连接程序(Linker)进行连接，最后生成可执行文件。每一步中出现了错误，都要倒回去，重来一遍。操作比较麻烦，这会给初学者带来一些困难。但是，这样生成的可执行文件，就可以在相应的操作系统环境下直接执行，不需要每次再去编译连接了。这样的可执行文件，已经脱离了原来的高级编程环境，成为可直接使用的软件。

QB 不同于早期的 BASIC，就在于 QB 也和其它的高级语言(如 Pascal、FORTRAN、C 等等)一样具有生成可执行文件的编译功能，适用于作软件的开发工具。它可以在 QB 集成环境下通过很简单的画面操作就可以产生可执行文件，也可以在 DOS 环境下进行可执行文件的生成。下一章将专门介绍。

4.2 如何调试 QB 程序

我们给电脑编写的程序从功能需求、算法确立、流程图编制、源程序代码的书写到上机输入、编译、执行，都要经过反复的修改、排错，才能生成一个好的程序。这个修改排错的工作，就是调试，英语中的 DEBUG，也是这个意思。

QB 提供了丰富的调试功能，有简便的自动语法检查、出错信息显示、单步跟踪执行、设置监视点和监视窗口、设置断点、自动跟踪等，这一节，首先介绍简单的调试功能，以下

将逐步介绍各个调试工具。

4.2.1　程序输入时的常见错误

● 输入语句中分隔符不符合规定

例如：在一行内输入两个赋值语句，必须用冒号(:)，如果输成逗号(,)

a=6,b=7

按回车键后，显示图 4－2 的错误信息。

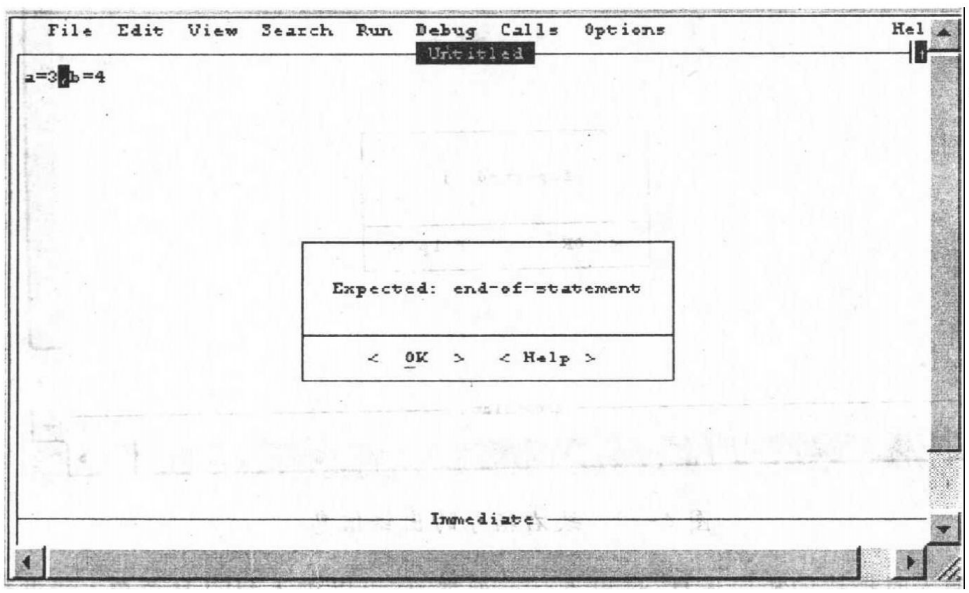


图 4－2　end－of－statement 错误信息

end－of－statement 表示语句中分隔符不正确。这是初学时最常犯的错误，其它还有一些语句输入时，该留空格的位置没有留，也会出现这种错误，比如：

IF X>0 THEN	输成	IFX>0THEN	(IF 与关系表达式之间、表达式与 THEN 之间必须留空格)
FOR N=1 TO 8	输成	FORN=1TO8	(FOR 与循环变量之间、初值与 TO 之间、终值与步长之间必须留空格)
M=15 MOD 16	输成	M=15MOD16	(MOD 运算的两个操作数与 MOD 之间 必须留空格)

总之，语句中的分隔符不正确时，都会产生这种错误。又如

INPUT X,Y	输成	TNPUT X;Y	或输成	INPUTX,Y
READ A,B,C	输成	READ A、B、C	或输成	READA,B,C

● 表达式中的括号不配对

例如：X=(−B+SQR(B * B−4 * A * C))/(2 * A)

上式中有三个左括号，就应有三个右括号，但是电脑发现你只输入了两个右括号，所以显示图 4−3 所示的出错信息。

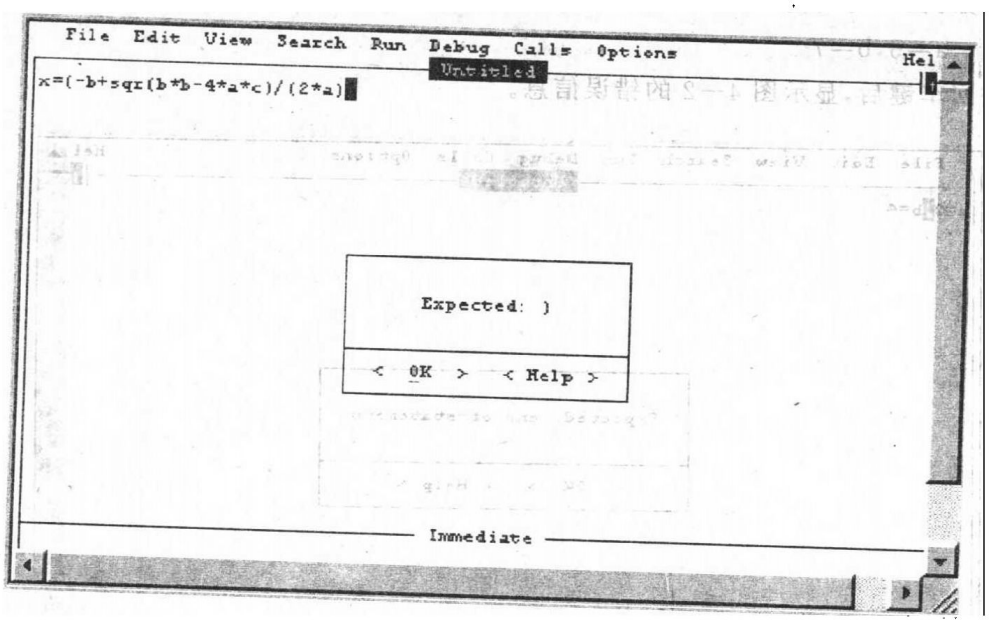


图 4−3　缺右括号的出错信息

其它输入时报错的相应信息，这里不一一列举，你可以进入 HELP 菜单，请求帮助。

以上错误在语句输入完回车就报错，你必须查明原因，当即改正，才能继续输入。然而，还有一些输入时错，语法检查也查不出来，一直要到执行时才报错。

4.2.2　程序执行时检查到的错误

● 语法错 Syntax error

1. 语句的关键字不正确，例如

INPUT	输成	IMPUT	(字母 M 应为 N)
FOR	输成	F0R	(字母 O 误为数字 0)
ELSE	输成	LESE	

2. DATA 语句中的数据类型与 READ 语句中的变量类型不一致。

例如：READ X,Y

```
S=X+Y
PRINT S
DATA 5,"8"
END
```

错误信息见图 4－4。

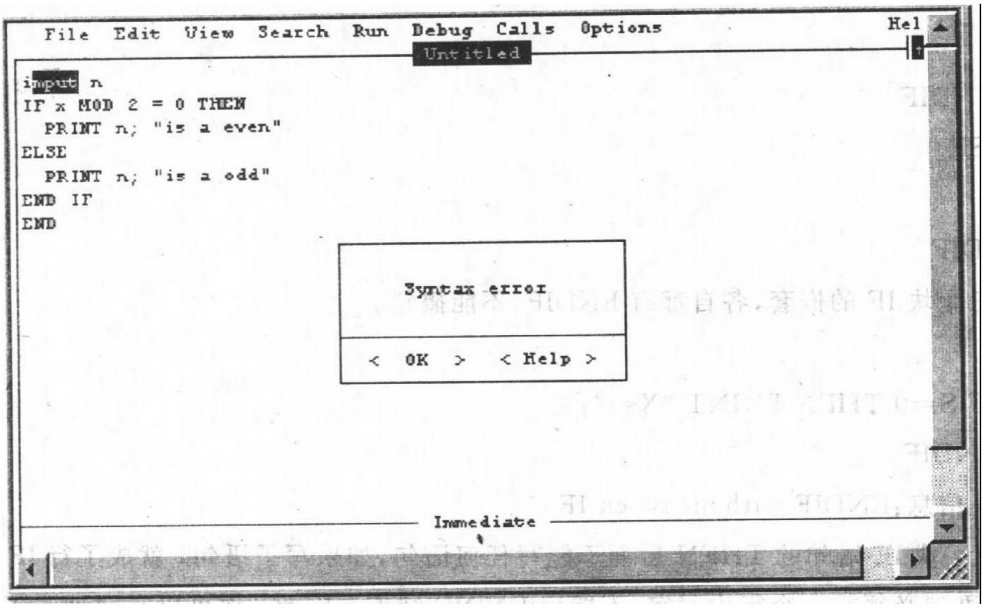


图 4－4　Syntax error 出错信息

产生 Syntax error 错的原因还有很多，要根据具体情况去查找分析。

● 块 IF 的结构不配对

1. IF Y<4 THEN
   X=Y * Y
   ELSE
   X=Y
   PRINT X
   END

错误信息：Block IF without ENDIF。

含义：块 IF 语句没有 ENDIF，正确的语法规定，一个块 IF，必须有一个 ENDIF 作为该块的结束标志，特别是在嵌套的情况下，更加需要注意。比如：

```
IF <  > THEN
  IF <  > THEN
```

```
        IF < > THEN
            ...
        ELSE
            ...
        ENDIF
    ELSE
      ...
    ENDIF
ELSE
  ...
ENDIF
```

有三个块 IF 的嵌套，各自都有 ENDIF，不能搞错。

```
2. IF S=0 THEN PRINT "X=";X
  ENDIF
```

错误信息：ENDIF without Block IF。

含义：块 IF 语句的 THEN 后面不能写任何语句，如果写了语句，就成了行 IF 语句，而行 IF 语句必须在一个行内写完，不能用 ENDIF 结束。因此，你可以有两种改正办法：把 THEN 后的语句搬到下一行去，构成块 IF；或者删去 ENDIF，保留 THEN 后的语句，成为行 IF 语句。这两种 IF 语句的用法，详见第七章。

● FOR－NEXT 循环语句不配对

```
1. FOR I=1 TO 9
    FOR J=1 TO I
      PRINT I;" * ";J;"=";I * J
    NEXT J
```

错误信息：FOR without NEXT。

含义：FOR 循环语句少了配对的 NEXT 语句，这在多重循环时可能发生错误。

```
2. FOR N=1 TO 5
  S=S+N
  NEXT I
```

错误信息：NEXT without FOR。

含义：FOR 循环的终端语句 NEXT 中的变量名必须与该循环的循环变量名相同，系统作语法检查时，首先看到 FOR，它就要去找相应的 NEXT，而找到该 NEXT，发现不是同一循环变量，但是它又不能识别后续是否还有 NEXT，所以它认为该 NEXT 没有配对的 FOR。

● 数据类型不匹配

```
DEFINT A－Z
  …
CALL SUB1(G,H)
  …
DEFDBL A－Z
SUB SUB1(A,B)
  …
END SUB
```

错误信息：Parameter type mismatch。

含义：参数的类型不匹配。在主程序调用语句中的实参与子程序的形参结合时一定要类型相同，上面的形实结合中，实参 G、H 是整型，而形参 A、B 是双精度型，所以，就出现参数类型不匹配的错误。

文件存盘时的一个错误

如果把贴有写保护的软磁盘放到驱动器中，当你用 Save 等命令存盘时，也会出现如下错误信息：

Permission denied

含义是：存盘的请求被拒绝，这是由于贴有写保护引起不能存盘的缘故。

4.2.3　如何查看错误信息

1. 查看用户手册或有关的技术资料

例如本书在附录中收入 QB 执行时的出错信息，按错误代码顺序排列，并有中文解释。

2. 查看 HELP 帮助

当信息框出现后，你按 ESC 键，信息框就消失了，但是光标仍停留在错误所在行，再按 F1 键，就会显示帮助信息，如图 4－5，它将简要地说明语句的语法或产生错误的可能原因。

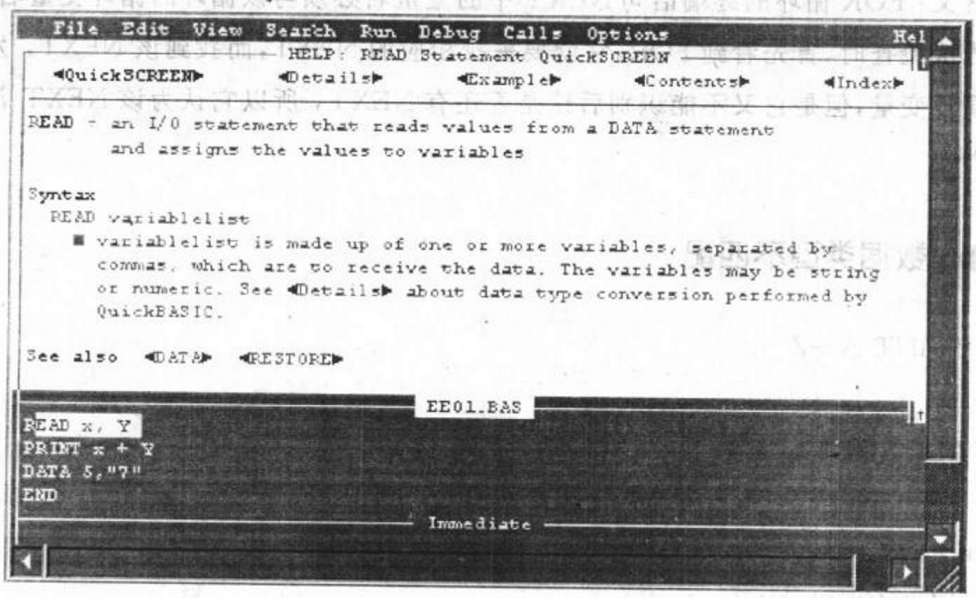


图 4—5　帮助信息画面

4.3　利用立即窗口进行调试

4.3.1　立即窗口(Immediate)的特点

按 F6 键就可以把光标从观察窗口(View)移动到立即窗口(活动窗口)，在这个窗口里键入语句，再按回车，语句就可以立即执行，如图 4—6，比如输入

print 56 * 12（回车）

屏幕马上转到 DOS 的结果画面，看到运算的结果。

可以用问号？代替关键字 PRINT，例如 ？56 * 12，和上面的语句完全等效。

因此，我们可以利用它把程序中的一些简单成份，边输入边执行，这样可以减少整体程序的调试工作量和出错机会。

用立即窗口进行调试，具有以下特点：

·立即窗口最多可输入 10 行，每输入一行，文本就向上滚动一行；

·每一行最多可有 256 个字符，因此，在一行里可以输入多个语句，各个语句之间要用冒号:分隔，例如：

FOR I=1 TO 5 :S=S+I :NEXT I 回车就执行这个循环，并立即看到结果；

·各行的语句是互不相干的，你可以把光标移动到任意一行去执行那一行的程序；

·立即窗口里的程序不能被存放到磁盘上。

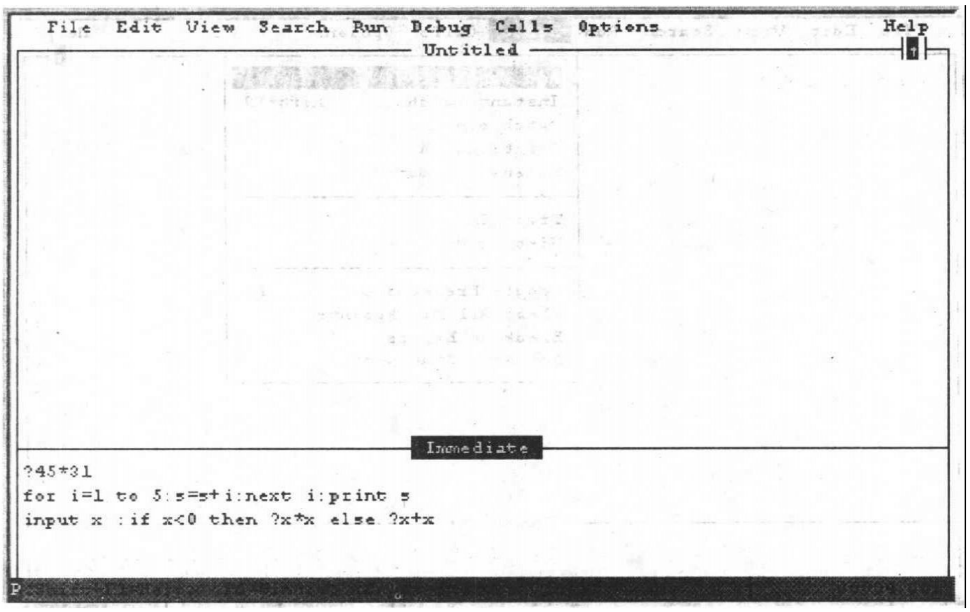

File Edit View Search Run Debug Calls Options Help
Untitled
Immediate
?45*31
for i=1 to 5:s=s+i:next i:print s
input x :if x<0 then ?x*x else ?x+x

图 4—6　立即窗口的输入

4.3.2　立即窗口允许的语句

立即窗口只能允许 QB 的部分语句在里面执行，下列语句不能在立即窗口执行：

COMMAND	ELSEIF	OPTION
CONST	END DEF	REDIM
DATA	END FUNCTION	SHARED
DECLARE	END IF *	$STATIC
DEF FN	END SUB	STATIC
DEFtype	END TYPE	SUB
DIM	FUNCTION	TYPE
$DYNAMIC	$INCLUDE	

块 IF 不能使用，但是行 IF 可以使用，例如：

IF X>0 THEN S=S+X ELSE S=S * X

4.4　QB 的调试功能

4.4.1　调试菜单

选择菜单条的 Debug 菜单，将显示如图 4—7 画面。

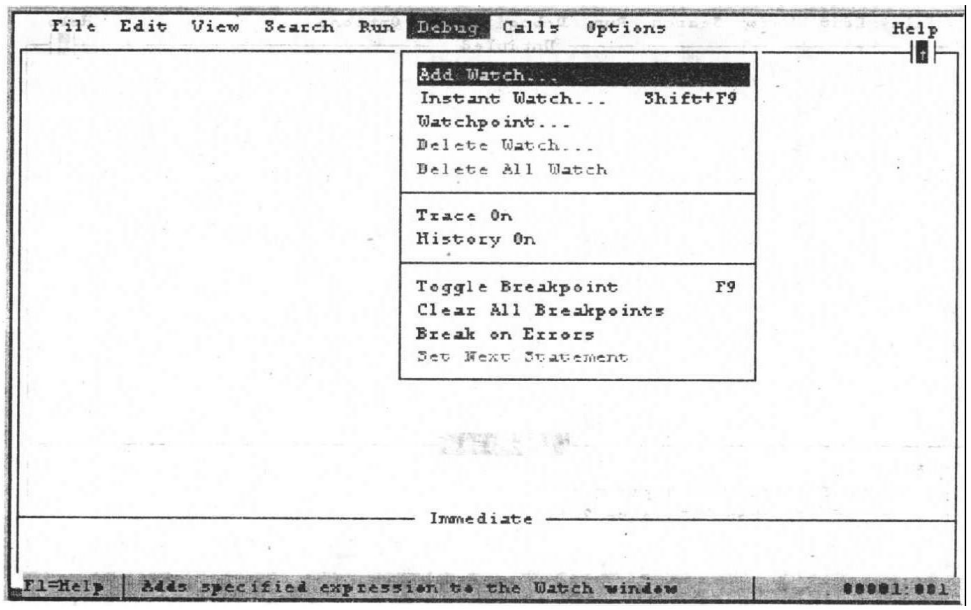


图 4—7　Debug 菜单画面

其中各个命令的意义如下：

Add Watch	设置监视表达式
Instant Watch	设置直接监视表达式
Watchpoint	设置监视点
Delete Watch	删除单个监视表达式或监视点
Delete All Watch	删除所有的监视表达式或监视点
Trace On	打开或关闭跟踪
History On	打开或关闭跟踪前 20 行程序
Toggle Breakpoint	设置断点
Clear All Breakpoint	取消全部断点
Break On Errors	设置出错中断
Set Next Statement	设置跳过一段程序的下一个语句

4.4.2　调试命令的功能

● 监视表达式

你可以观察程序执行期间某些变量或表达式的值，当你设置后，就会在屏幕上方出现一个监视窗口，如图 4—8 所示。

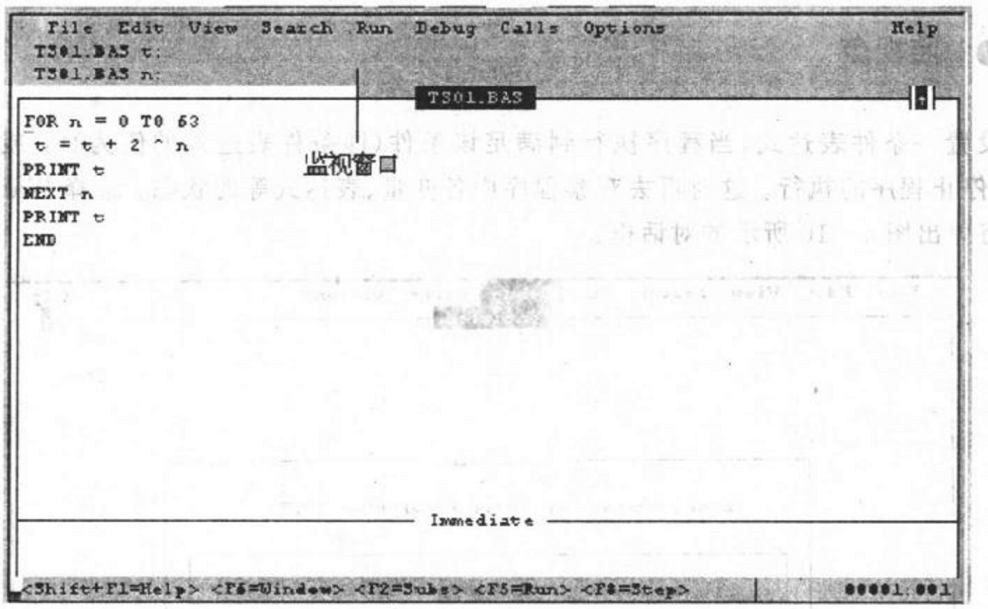


图 4—8　监视窗口

只要一执行程序，该窗口中设置的监视表达式的值就会显示出来。

选 Add Watch 命令，每次只能设置一个变量或表达式，如果要想观察多个变量或表达式，可以多次设置。

当不需要再观察某一变量或表达式，可用 Delete Watch 命令取消，选此命令后，出现如图 4—9 所示的对话框。

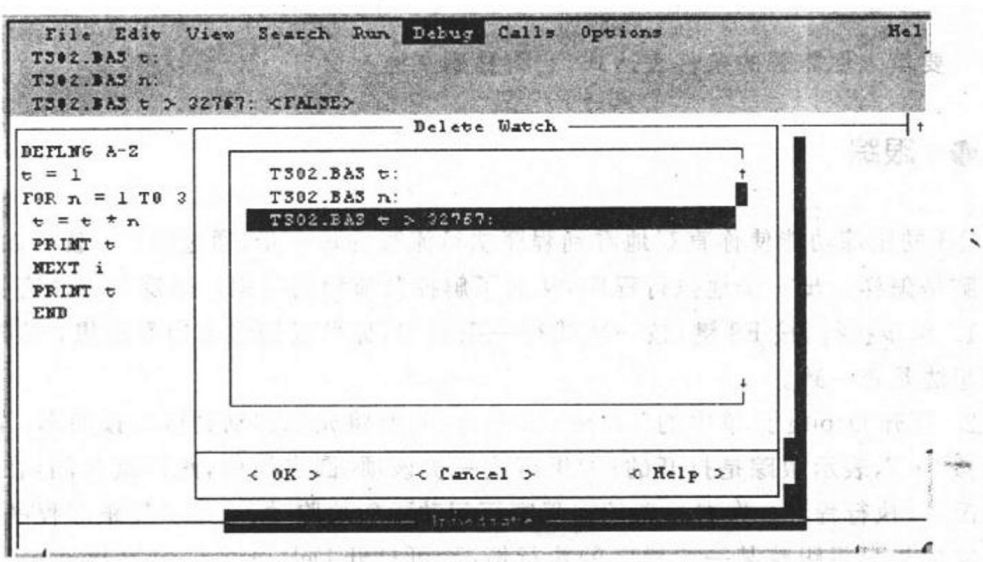


图 4—9　Delete Watch 命令的对话框

用光标选择你要取消的项目，按回车，该项将从监视窗口消失；

如果你想全部取消监视表达式，则选 Delete All Watch。

● 监视点

设置一条件表达式，当程序执行到满足该条件(即条件表达式的值为真，或非 0 的值)就停止程序的执行。这时再去观察程序中各变量、表达式等的状态。选择 Watchpoint 命令将弹出图 4—10 所示的对话框：

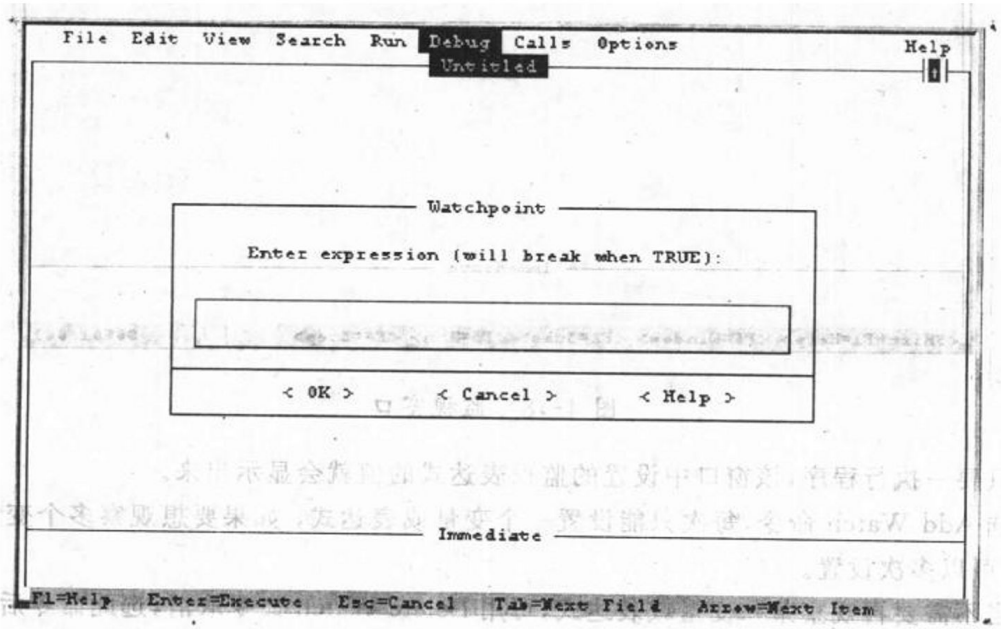


图 4—10　Watchpoint 命令的对话框

只要填入你需要的条件表达式，它将显示在监视窗口。

● 跟踪

QB 的跟踪功能使你直观地看到程序执行流程的每一步，通过跟踪，就可以更好地了解电脑是怎样一步一步地执行程序，从而了解控制流程的过程。跟踪的方法有：

1. 单步执行，按 F8 键，按一次执行一条语句，你可以慢慢地边看边想，这样设计，和你的想法是否一致。

2. 打开 Debug 菜单中的 Trace On 命令，只要将光标移动到该项按回车，该项 前有一圆点“·”，表示跟踪是打开的；如果再按一次该项，圆点消失，跟踪就关闭，在跟踪打开的情况下，执行程序，将有一亮条在程序行间动画般地跑动，这是快速跟踪程序执行。

3. 如果只想跟踪某一段程序的执行情况，可打开 Debug 菜单中的 History On 命令，方法与打开 Trace On 相同。但是先要把光标移动到该程序段的最末一行，它将来回跟 踪前 20 行程序的执行。

● 断点

程序暂停执行的位置，就叫断点。你想程序执行到某一处就停止，以便查看已执行部分的情况，因此，先将光标移动到预定的位置，再选 Debug 菜单里的 Toggle Breakpoint 命令，断点就设置好了。当你再去执行程序时，程序将在断点处停下来，你就可以查看各种情况。

根据需要你可以在一个程序中设置多个断点，而当你不需要断点时，用 Debug 菜单里的 Clear All Breakpoint 命令清除所有断点。

4.5 程序调试实例

4.5.1 了解分支程序的执行过程

例：从键盘输入一个数，判断是奇数(Odd)还是偶数(Even)

【源程序】

```
'Example QB4—1.BAS
1 INPUT n
2 IF n MOD 2 = 0 THEN
3   PRINT n; "is a even"
5 ELSE
6   PRINT n; "is a odd"
7 ENDIF
8 END
```

【说明】

1. 判断数的奇偶，用 2 去除它，如果能整除，余数为 0，是偶数；反之，不能整除，余数不为 0，是奇数。QB 有一个求余运算符 MOD，两个操作数写在它的两边，例如

X MOD Y

X 是被除数，Y 是除数，X\Y(整除)其余数是 MOD 的值，比如 15 MOD 16，其值为 15，8 MOD 2 =0。

2. IF—THEN—ELSE 是两分支的选择结构语句，当条件表达式的值为真，表示条件成立，执行 THEN 后的语句，直到 ELSE，条件表达式的值为假，表示条件不成立，执行 ELSE 后的语句，直到 ENDIF。执行流程如图 4—11 所示。

3. 程序中每行前的行号是为便于说明流程而加的，实际运行程序可以不要。

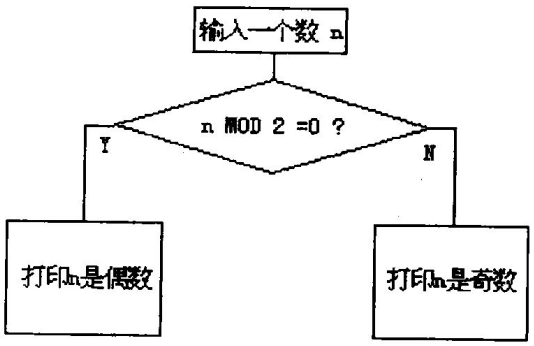


图 4—11　分支流程图

【程序调试】

1. 用 Debug 菜单里的 Add Watch 命令，设置监视表达式，输入变量 n，于是屏幕上方出现监视窗口。

2. 用 F8 单步执行，从第 1 行开始，按 F8 键，执行 INPUT 语句，从键盘输入 n，画面转到状态 DOS，这时输入一个数 56，回车后，画面又返回窗口画面，第 2 行出现反白亮条，监视窗口中的 n 值，已经变为输入的值，如图 4—12 所示；

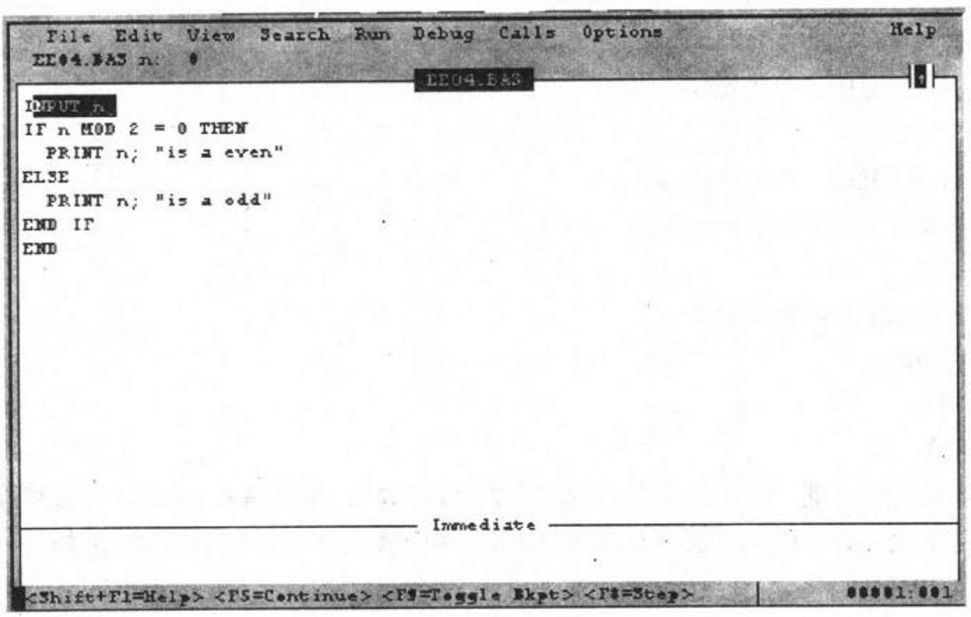


图 4—12　单步跟踪开始画面

3. 再按 F8，反白亮条移动到第 3 行，这是因为 56 MOD 2 的值为 0，条件成立，执行 THEN 后的语句，第 3 行是 PRINT 语句，输出结果到屏幕，你可以观察到屏幕闪动一下，反白亮条跳过第 5、6 行，转到第 7 行，再到最末的第 8 行，程序结束，如图 4—13 所示。

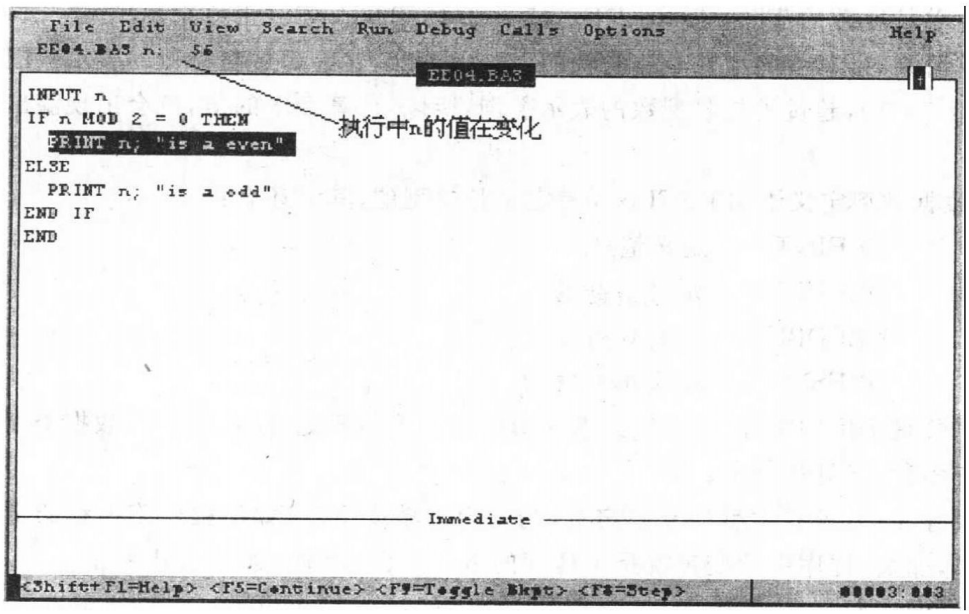


图 4—13　单步跟踪结束画面

4. 总之，条件成立的流程为 1、2、3、7、8

条件不成立的流程为 1、2、5、6、7、8。

4.5.2　一个有溢出(Overflow)错误信息的程序调试

例：计算阶乘 n!（设 n=20）

【源程序】

```
'Example QB4—2.BAS
DEFLNG A—Z          '定义长整型变量
t = 1
FOR n = 1 TO 20
  t = t * n
  PRINT t
EXIT n
PRINT t
END
```

【程序说明】

1. 执行这个程序将出现错误信息：Overflow(溢出)。

错误的原因是变量的类型定义为长整型，在电脑中，不同类型的数据，允许数值的范围也不一样，比如：

整型数的范围：—32768～+32767

长整型数的范围：—2147483648～+2147483647

单精度数的范围：1.40129E－45 ～3.402823E＋38(正数)

计算阶乘时，其值的增长是非常快的，8!＝40320，这已超过整型数的表示范围，13!＝6.22702＊10^9，超过了长整型数的表示范围，所以，计算 20! 的值，就会出现溢出错误信息。

2. 数据类型定义语句 DEFLNG 是定义长整型的，其它还有：

DEFINT	定义整型
DEFSTR	定义字符型
DEFDBL	定义双精度型
DEFSNG	定义单精度型

这些关键字后的字母，表示以这些字母开头的变量都定义为该类型。数据类型的定义方法还有多种，详见第七章。

如果程序中，没有类型的专门定义，约定为单精度型，假如把程序第一行的定义类型语句删去，那么，程序中的变量就是单精度的了，再次执行时，就不会出错了。

【程序调试】

1. 用 Debug 菜单里的 Add Watch 命令设置两个监视表达式：n 和 t(要分两次输入：用 Watchpoint 命令设置一个监视点，观察当 n 大于 13 时的情况，设置后的监视窗口；同时，也将跟踪打开，如图 4－14 所示。)

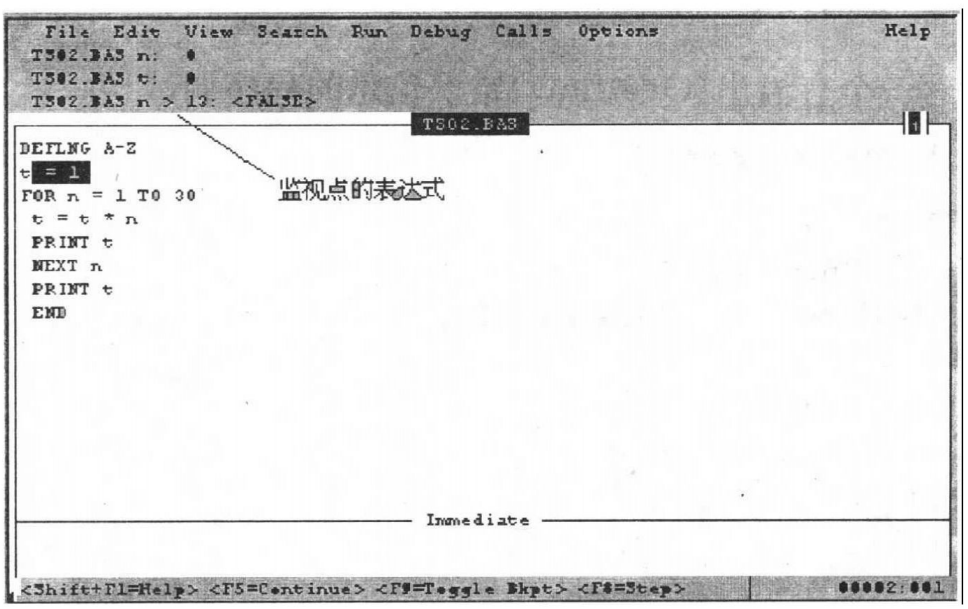


图 4－14　执行前的监视窗口

2. 用 Run 菜单中的 Start 命令，执行程序(也可用你已经用得很熟的命令去执行程序)，这时注意观察屏幕上亮条的闪动，表示程序在跟踪执行，不过，由于程序很短，一瞬就完了，程序已经暂停，看看监视窗口中各项的变化，如图 4－15 所示，这时 n 是 13，大于 12，监视点的表达式为真(屏幕上显示 True)，因此，执行暂停。这时 t＝479001600，这是 12! 的值，下一次执行 13! 已经超过长整型的表示范围，因而，一个出错信息框也出来了。

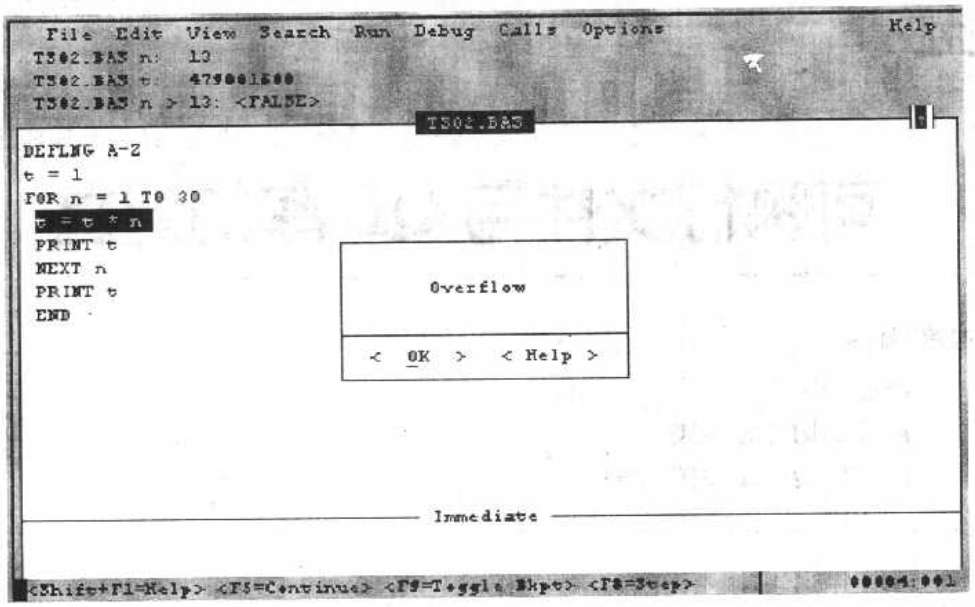


图 4—15　监视点的值为真时程序暂停的画面

3. 这时程序处于暂停状态，还没有执行完，你可以用 Run 菜单里的 Reset 命令使程序复位，回到开始状态，监视的值也都回到初值。

第五章

可执行文件与QB库的建立

本章内容：

- ▶在QB环境下建立可执行文件
- ▶库的功能和种类
- ▶建立Quick库的方法

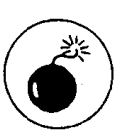
5.1 建立可执行文件

本章之前我们建立的程序都要在QB的集成环境下才能执行，存放的文件名是带扩展名.BAS；现在我们要建立一个不需要QB集成环境支持，而在DOS下直接运行的程序文件，这种文件叫做可执行文件，其扩展名为.EXE。在DOS状态下，只要键入可执行文件的名字(不要后缀.EXE)，就能立即执行该程序。

建立可执行文件有两种不同的途径：一是在QB的画面上操作；一是在DOS状态下进行编译和连接。本章介绍前一方法，后一方法在下一章介绍。

5.1.1 在QB环境建立可执行文件

假定你已经在内存中装入一个程序，而且是已经运行通过了的，则可通过下列步骤建立可执行程序。



不要在没有装入程序或程序还没有调试通过的情况下做下面的事。

1. 选择菜单命令

选择Run菜单(图5—1)中的Make EXE File命令，其对话框如图5—2所示。

2. 确认文件名

在EXE File Name框中显示出可执行文件的名字，如果不想使用这个名字 可以键入另外的文件名(不带扩展名)；否则进行下一步。

3. 打开或关闭查错功能

将光标移动到[ ]Produce Debug Code栏按回车键，方括号内出现一个标志“*”查

错功能打开，程序执行时将进行如下检查：

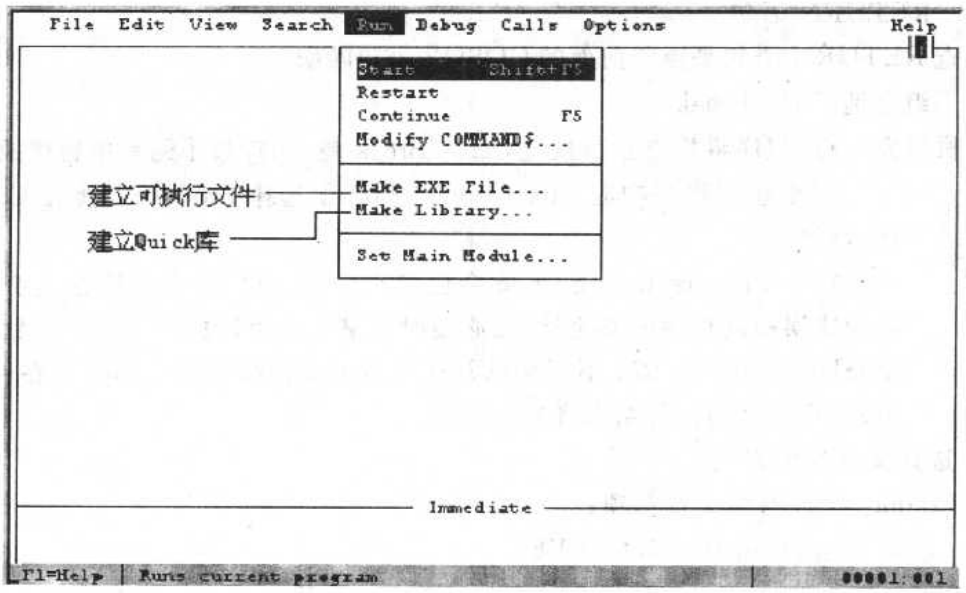


图 5—1　Run 菜单画面

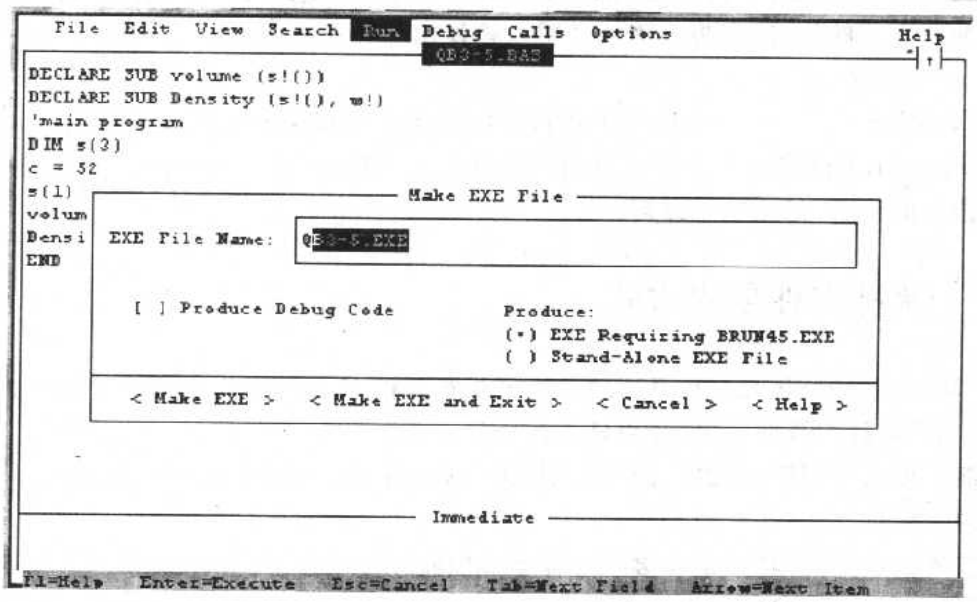


图 5—2　Make EXE File 命令的对话框

□　算术溢出

所有的算术运算，包括整数运算和浮点运算，都要进行溢出检查(上溢和下溢)。

□　数组界限

检查数组下标，看它是否在由 DIM 语句说明的范围内。

□　行位置

可以指出发生错误的行。

□ RETURN 语句

检查 RETURN 语句是否与前面的 GOSUB 语句匹配。

□ 组合键 Ctrl+Break

每执行完一行，程序将检查是否按过 Ctrl+Break 键，如按过了则程序暂停执行。

如果想让程序响应 Ctrl+Break，则必须选择 Produce Debug Code 来编译、连接程序。

· 打开 Produce Debug Code 会使可执行文件增大，降低运行速度，但为了减少错误提高程序的安全性，在必要时还是要使用该选择项。否则，数组超界、GOSUB 与 RETURN 不匹配以及算术溢出等类型的错误都不能在错误信息中显示，程序的执行结果将难以预料。

4. 选择文件生成方式

在 Produce 下面有两个选择项：

□ EXE Requiring BRUN45.EXE

建立的可执行文件必须同时与 BRUN45.EXE(运行模块)一起，才能执行。

□ Stand—Alone EXE File

建立的可执行文件不需要与 BRUN45.EXE(运行模块)一起，就能够执行。

这两种方式的优、缺点将在稍后说明。建议初学者选第二项。

5. 选择执行

Make EXE	建立可执行文件后返回到 QB 环境。
Make EXE and Exit	建立可执行文件后返回到 DOS 提示符。

根据你的需要，任选其一即可。

5.1.2 两种文件生成方式

1. BRUN45.EXE 是 QB 系统的一个可执行文件，我们称为运行模块，它主要包含有增强 BASIC 语言所需的程序，使用该模块的优点是：

· 缩短可执行程序的长度，因为放到 BRUN45.EXE 中的子程序模块，就不需要再放到新建的文件里。

· 节省磁盘空间，特别是有多个这种文件同时存盘的情况，节省更多。

· 可以保护 COMMON 语句中的变量。

· 当 BRUB45.EXE 已装入内存时，就无需再装入连接程序系统中的每个程序。

2. 建立独立程序(Stand—Alone EXE File)，不需要运行模块。它有两个优点：

· 如果程序较小而且简单，即不需要访问运行模块中的多个子程序，在运行时可节省内存空间。

· 使用方便，因为程序在运行时不需要运行模块在盘上。这在编写需要用户拷贝的程序时是很有用的。

5.1.3　建立可执行文件所需的程序

为了建立可执行文件，下列四个文件必须在当前目录或由PATH命令所指定的有效目录下。

1. BC.EXE

这是一个编译程序，它把源程序(.BAS文件)编译成可供连接的目标文件(.OBJ文件)。

2. LINK.EXE

这是一个连接程序，它连接由BC建立的目标文件。

3. BRUN45.LIB

这是一个子程序库，为BASIC程序执行诸如接收输入或在屏幕上输出文本这样的操作提供支持。这些子程序通过LINK.EXE与目标文件相连接。BRUN45.LIB被称为运行模块库，当一个程序与该库相连接生成可执行文件后，执行这个可执行程序时，BRUN45.EXE必须存在。

4. BCOM45.LIB

这也是一个子程序库。其功能与BRUN45.LIB相同，不同的只是由该库建立的可执行文件能够独立运行，即不需要运行模块BRUN45.EXE。该库被称为交叉运行模块库。

5.2　Quick 库

象装积木游戏木块的盒子一样，QB中的模块化程序设计，把那些常用的或不同用途专用的子程序、函数(总称之为过程)装在一起，存放这些过程的文件就叫库，或库文件，也就是说，库是一种具有唯一文件名的含有若干个模块的特殊文件。库里的模块可以被其它程序调用。

库是QB的重要特征之一，利用它，不仅可以实现模块化程序设计，而且可以把其它语言的子程序放入库中供Quick BASIC程序调用。

5.2.1　库的分类

QB提供创建两类不同库的方法，创建的这两类库以不同的文件扩展名来区分。

1. Quick 库

以.QLB作扩展名。一个Quick库可以含有用Quick BASIC或其它Microsoft公司的语言(如FORTRAN、C等)编写的过程。利用这种库可以根据需要对QB系统进行扩充。

2. 独立库

以.LIB作扩展名。当QB建立一个Quick库时，同时也建立了一个独立库。两种库中

的过程相同，只是形式上略有差别。

当用QB装入库时，可以把Quick库看成一组“连接”到QB系统上的过程。独立库通常是编译过但未连接的过程，既可以把它们加到一个Quick库中，也可以与一个主模块相连接来建立一个在DOS命令状态下执行的文件。

5.2.2 Quick库的优点

1. Quick库利于程序的开发和维护

在开发程序的过程中，需要编写含有各种过程的模块。随着开发的进一步深入，一些模块渐渐成为程序的固定成分，于是就可以把它们加到一个Quick库中，将其作为原始模块的源文件“封装”起来直到你想对它们进行改进和维护为止。装进库之后，就可以将QB程序与库一起装入内存，程序就可以直接访问库中的所有过程。在程序中写一个过程名及实际参数就行了，Quick库中过程的作用如同QB的语句，你不必重复地写那些过程，省时省力，大大提高编程的效率。

2. Quick库中可以含有其它语言写的子程序。可以充分利用各种语言的特点，增强程序的功能。

3. Quick库特别适合于开发大型程序。

5.2.3 Quick库的建立

Quick库的建立既可以在QB环境下进行，也可以在DOS状态下进行。本章先介绍在QB集成环境下的建库方法。

● 建库前应了解的事项

1. 当建立一个新的Quick库时，该库自动包含QB环境中现存的所有模块，以及启动QB时装入的其它Quick库的内容。

2. 如果装入了一个完整的程序，只想把某些模块放入库中，可以用File菜单中的Unload File命令来删除那些不需要的模块。用View菜单的SUBs命令可以查看装入了那些模块。但这种方法无法显示一个库中含有哪些过程。

3. 在库中只能放入完整的模块。如果一个Quick库只需要一个模块中的某些过程，则应把所需要的过程放进一个独立的模块，再将该模块放入库中。

4. Quick库必须自成一体，因此，被Quick库中的某个过程调用的所有过程也必须在该库中。

5. 建立Quick库需要以下文件在当前目录或在DOS环境变量所指定的路径中：

QB.EXE	QB集成环境下控制建库过程
BC.EXE	将源程序翻译成目标代码程序
LINK.EXE	连接目标文件

LIB.EXE	管理目标模块的独立库
BQLB45.LIB	提供QQuick库所需的子程序

● 建库的操作步骤

1. 选择画面

在装入Quick库需要的所有模块后，选择Run菜单的Make Library命令，这个命令执行后出现如图5－3所示对话框。

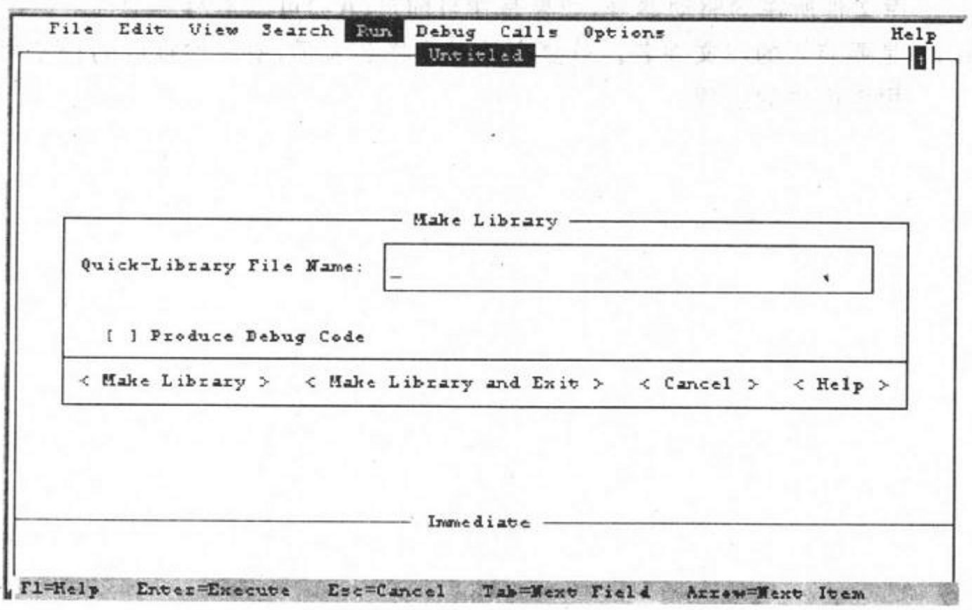


图5－3　Make Library对话框

2. 输入库的名字

在Quick－Library File Name栏内输入要建立的Quick库文件名。如果只输入基本名，则系统自动加上扩展名.QLB；如果不要扩展名，可在输入基本名后加一个句点“.”。

3. 选择查错功能

如果想检查正在建库中的模块里存在的错误，可选择Produce Debug Code框。但这样将导致库增大，程序执行速度减慢，并且只给出少量的错误控制。这种选择主要用于对数组上下界的检查。在一般情况，建议你不选该项。

4. 执行方式选择

Make Library	确定建库但不退出QB环境
Make Library and Exit	建库并退出到DOS环境
Cancel	不建库
Help	请求系统提供帮助信息

5.2.4 如何使用 Quick 库

当你把 Quick 库建好以后，你的磁盘上有一个库文件名，其扩展名为.QLB，比如假定有这样一个库 Math.QLB，调用格式如下：

A> QB/L B:Math（回车）

其中参数的意义是：

/L　告诉计算机装入库

B:　库文件所在的驱动器号，如果是在当前盘(A:)可以不写

Math　是要装入的库文件名。只要启动时这样装入了，就可在以后的程序中直接使用该库中的过程。

第六章

在DOS环境下建立可执行文件

本章内容：

▶在 DOS 环境下使用 BC 命令编译源程序

▶使用 LINK 命令连接目标文件

QB 除了能在其编程环境下利用菜单命令(Run 菜单的 Make EXE File 命令)来生成可执行文件外，也可以在 DOS 状态下使用 BC 命令(编译)和 LINK 命令(连接)来生成可执行文件，且有很多优点。

1. 充分利用内存资源。对在 QB 环境下内存空间不够的程序特别有用。

2. 可以使用不同的文本编辑器。

3. 产生的可执行程序可由 Microsoft Code View 调试程序进行动态调试。

4. 可以产生列表文件，用来调试独立的可执行程序。

5. 可以使用很多选择项来实现指定的操作。

6.1 编译和连接的一般过程

1. 编译源文件，建立一个目标文件。

2. 把这个目标文件与一个或更多的独立库连接，以便建立一个可执行文件。

连接程序建立可执行文件之前，必须确认所有源文件中的过程与被调用库中的过程或其 它目标文件中的过程的参数要匹配。(即形参和实参的个数相等、位置一一对应、类型相同)

用户可以用以下两种方法之一来进行编译和连接：

1. 分别使用 BC 和 LINK 命令来独立地进行编译和连接。

2. 建立一个批文件，该文件包含所有的编译和连接命令。这时在编译和连接时使用的选择项是相同的。

6.2 用 BC 命令编译

6.2.1 操作过程

可以用以下两种方法之一来执行 BC 命令。

方法一：在 DOS 状态的提示符下键入 BC，按回车键后出现提示信息，如图 6－1 所示。

```
C:\CL\QB45>BC
Microsoft (R) QuickBASIC Compiler Version 4.50
(C) Copyright Microsoft Corporation 1982－1988
All rights reserved.
Simultaneously published in the U.S. and Canada.
Source Filename [.BAS]:←——输入源程序的基本名
```

图 6－1　BC 命令的提示信息

在提示信息中，你要输入的内容如下：

Source Filename 〔.BAS〕：输入源程序文件的基本名，回车

Object Filename 〔basename.OBJ〕：输入目标文件的基本名，回车

Source Listing 〔NUL.LST〕：输入列表文件的基本名，回车

其中方括号内表示默认的扩展名，所以你就不需要输入扩展名了。

源程序文件的基本名就是你已经用编辑程序编辑好了的、存放在当前磁盘上的文件的名字；如果不是当前盘和当前路径，必须还要指定文件的盘符和路径。

目标文件名和列表文件名可以不输入（第一项的源文件名必须输入），只按回车键，系统按缺省值处理，自动以源文件的基本名作这些文件的基本名。

以上信息应答后，电脑开始编译，如果没有出错，将出现以下信息：

```
C:\CL\QB45>BC
Microsoft (R) QuickBASIC Compiler Version 4.50
(C) Copyright Microsoft Corporation 1982－1988
All rights reserved.
Simultaneously published in the U.S. and Canada.
Source Filename [.BAS]:b:qb3－5
Object Filename [QB3－5.OBJ]:
Source Listing [NUL.LST]:

43885 Bytes Available
42882 Bytes Free

     0 Warning Error(s)——警告性错误
     0 Severe Error(s)——严重性错误
C:\CL\QB45>
```

图 6－2　正确编译完成后的信息

如果在编译中出现了错误，它将显示如下的信息提示：

XX Warnning error　　表示有多少个一般警告性的错误

XX Sever error　　表示有多少个严重性的错误

另外还有出现错误的位置和错误信息，请参考QB出错信息表。

这时又要回到文本编辑程序，修改错误，然后再次用BC编译，一直到全部正确为止。

```
C:\CL\QB45>BC
Microsoft (R) QuickBASIC Compiler Version 4.50
(C) Copyright Microsoft Corporation 1982-1988
All rights reserved.
Simultaneously published in the U.S. and Canada.
Source Filename [.BAS]:b:TS02
Object Filename [TS02.OBJ]:
Source Listing [NUL.LST]:

0044 000A NEXT i
                ^ NEXT without FOR←——      出错信息
0084 000E
                ^ FOR without NEXT←——

43885 Bytes Available
43553 Bytes Free

    0 Warning Error(s)
    2 Severe Error(s)——有两个严重性错误

C:\CL\QB45
```

图6—3　编译出现错误后的信息

方法二：在单一命令行上键入编译需要的所有信息，格式为：

BC 源文件名〔,〔目标文件,列表文件,选择项〕〕〔;〕

格式中的源文件名不可少，其它项可选。如果在源文件后跟一个分号，则表示后面的可选项缺省。

例如：C> BC QB3—5;

回车后就立即编译，无需应答上述信息。

6.2.2　BC命令的选择项

使用BC命令进行编译时，可以带一个或多个选择项，每个选择项以斜线(/)或连字符(—)开头，后跟一个或多个字母。

现将这些选择项介绍如下。

□　/a

为每个源程序行生成分解目标代码的列表；显示由编译程序生成的汇编语言代码。

□　/ah

允许记录的动态数组、定长字符串和数值数据占用所有的可用内存空间。该选择项缺省时，每一数组的最大长度为64KB。这个选择项对过程之间数据的传递方式没有影响。

□ /c:buffersize

为接收远程数据设置缓冲区容量，远程数据由异步通信适配器从每个通信口接收而来(分配给每个通信口的传输缓冲区为128字节，在BC命令行中不能改变)。如果没有异步通信口，此选择项无效。两个口的缓冲区缺省总容量为256字节，最大为32767字节。

□ /d

生成作为运行过程中错误检测的调试码，并且使Ctrl+Break键有效。它类似于QB环境中Run菜单的Make EXE File命令的Produce Debug Code选择项。

□ /e

指明在源文件中带有RESUME行号的ON ERROR语句。

□ /mbf

使QB的转换函数把IEEE格式数作为Microsoft二进制格式数来处理。把内部函数MKS $、MKD $、CVS、CVD相应地转换为MKSMSF $、MKDMBF $、CVSMSF、CVDMBF。

□ /o

用BCOM45.LIB运行的库代替BRUN45.LIB。

□ /r

按行序存储数组。QB通常按列序存储数组。如果要使用按行序存储数组的其它语言的子程序，则该选择项会很有用。

□ /s

把引号内的字符串写入.OBJ文件，而不是写入符号表。在一个有许多字符串常数的程序里，当出现"Out of memory"错误时，使用此选择项。

□ /v

可以对下列事件产生陷阱：通信(COM)，光笔(PEN)，操纵杆(STRIG)，计时器(TIMER)，音乐缓冲区(PLAY)，功能键(KEY)。利用这个选择项，可检测语句间发生的事件。

□ /w

与/v一样能产生事件陷阱，只是用来检测行之间发生的事件。

□ /x

指明在源文件中存在带RESUME、RESUME NEXT、RESUME0的ON ERROR语句。

□ /zd

产生一个与源文件行号对应的行号记录目标文件。如果用Microsoft Symbolic实用调试程序(SYMDEB)作源文件级的调试，则该选择项会很有用。SYMDEB在Microsoft宏汇编4.0中。

□ /zi

产生一个含有调试信息的目标文件，该文件可用Microsoft Code View调试程序进行调试。在Microsoft C 5.0和Microsoft宏汇编5.0中有Microsoft CodeView调试程序。

6.3 用 LINK 命令连接

程序编译完成后，必须用相应的库连接目标文件生成一个可执行文件。
使用 LINK 命令的方法有以下两种。

1. 键入 LINK，按回车键后出现如图 6—4 的提示：

```
C:\CL\QB45>Link

Microsoft (R) Qverlay Linker Version 3.59
Copyright (C) Microsoft Corp 1983—1988. All rights reserved.

Object Modules[.OBJ]:_←——输入被连接的模块的基本名
```

图 6—4　Link 的提示信息

出现提示信息后应输入的内容：

Object Modulis[.OBJ]：输入被连接的目标模块的基本名，然后回车。如：QB3—5。

Run File〔QB3—5.EXE〕：如果你确认用该名作可执行文件的基本名，回车即可，如果你想另外取名，可再输入一个名字。

List File〔NUL.MAP〕：如果想要建立映象文件，就输入一个基本名，如果不需要建立此文件，可直接回车。

Libraries〔.LIB〕：输入连接时要使用的库文件名，一般使用系统默认的库，只需按回车键就可以了。

一般情况，第一项的目标文件不能省，其它的运行文件名、映象文件名和库名都可以不输入，按默认处理，只须在该项按回车键即可。

连接成功后，出现如图 6—5 所示的画面：

```
C:\CL\QB45>Link

Microsoft (R) Overlay Linker Version 3.59
Copyright (C) Microsoft Corp 1983—1988 All rights reserved.

Object Modules[.OBJ]:qb3—5
Run File [QB3—5.EXE]:
List File [NUL.MAP]:
Libraries[.LIB]:

C:\CL\QB45
```

图 6—5　连接成功后的画面

这时，在 DOS 的提示符下，键入可执行文件的基本名回车，程序就被执行，比如

QB3－5 的执行结果就显示在画面上，如图 6－6 所示。

```
C:\CL\QB45>qb3-5

The volume is 54
The Density is 962963

C:\CL\QB45>_
```

图 6－6 在 DOS 下程序执行的结果

如果连接中出现错误，则连接失败，必须查明原因，回头再去编辑源程序，再次 编译，再次连接，直到完全正确。

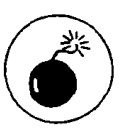
2. 在单一命令行上输入需要的所有信息，格式如下：

LINK 目标文件〔,〔执行文件〕〔,〔映象文件〕〔,〔库〕〕〕〔选择项〕〔;〕〕

格式中的目标文件名不可少，其它项可选。如果在目标文件后跟一个分号，则表示后面的可选项缺省. 例如：

C>LINK QB3－5；

这意味用 QB3－5 作可执行文件的基本名，不产生映象文件，用系统默认的库连接。



命令行的长度不得超过 128 个字符。

第七章

用QB语言编程

本章内容：

- ▶QB的基本语法知识
- ▶输入和输出语句：LET、INPUT、READ/DATA、PRINT
- ▶选择结构语句：块IF和SELECT CASE
- ▶循环结构语句：FOR和DO—LOOP
- ▶子程序和函数
- ▶数组的定义和应用
- ▶文件的打开与建立
- ▶字符处理、作图、音乐方面的初步应用

7.1 基本的语法知识

7.1.1 字符集

Quick BASIC字符集包括英文字母、数字字符和专用符号。

- 英文字母：大写A—Z，小写a—z
- 数字：0—9
- 专用符号(28个)：空格、＝、＋、－、＊、/、\、∧、(、)、〔、〕、%、#、$、!、&、，、．、'、；、：、?、<、>、"、—、@

7.1.2 模块、程序行、语句

● 模块

QB程序由一个或多个模块组成。每个模块是可以分别编译的一段程序，并以源文件形式独立存放在磁盘上。

多模块程序，必须有一个主模块(Main Module)，程序才是可执行的。在简单的情况下，程序只有一个主模块。

模块中的语句可以分为两级，在过程中的语句称为过程级代码，通常以 SUB 或 FUNCTION 开头，以 END SUB 或 END FUNCTION 结尾；在过程之外的语句称为模块级代码。主模块中的模块级代码通常称为主程序。主模块以 END 语句结束。

● 程序行

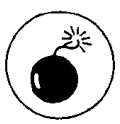
每个模块由若干程序行构成。一个程序行可分为行标识符、语句、注释三个部份。

1. 行标识符

QB 允许以下两种行标识符：

· 行号：从 0 到 65529 之间的无符号整数，但一般不要用 0 作行号，以免某些程序中产生混乱。

· 标号：由字母、数字组成，最多不超过 40 个字符，必须以字母开头并以冒号结束，不能与 QB 的保留字同名。例如 LABEL1：、LOOPN：、START：等等，都是有效的标号。标号中字母的大小写没有区别，例如 ALPHA：与 alpha：是同一个标号，不能作为两个标号。



在 QB 中，行号不决定程序执行的顺序，这一点与 GW BASIC 等较早的 BASIC 语言是不同的。

QB 中，程序执行的顺序是按程序行的排列位置进行的。为了改变顺序执行的流程，可以使用控制流程的语句，所以 QB 的行标识符主要用于流程控制中跳转的目标位置，不是每一个程序行所必须的。使用行标识符时，同一程序中，行号和标号可以混合使用，但一个程序行只能有一个行标识符。

2. 语句

(1) 语句是程序行的主体，是电脑具体操作的规定。它有可执行语句和非执行语句两类。

· 可执行语句

如 LET、PRINT 等，它是指定电脑的一种操作，源程序编译后，产生机器代码。

· 非执行语句

如 REM、DECLARE、DIM 等，它不指定电脑的具体操作，主要用于为变量分配存贮单元、定义变量类型等，为源程序编译提供信息，编译后不产生机器代码。

(2) 每一个语句可以分为语句定义符和语句体两部份。

· 语句定义符

这是每一个语句必不可少的，用以说明该语句的功能，通常用相应的英文词或英文词的缩写表示。如 READ、INPUT、IF THEN 等。

· 语句体

这是该语句的操作内容，包含了操作的对象和动作。有的语句不需要语句体，所以它不是每个语句所必须具有的。

(3) 在一个程序行中，可以包含多个语句，各语句之间用冒号“:”隔开，例如

A=8:B=13

(4) 一个程序行最多允许 256 个字符。

3. 注释

在程序行里，可以用一个注释语句 REM(或者用一个单引号“'”)引出说明该语句的内容，以便阅读理解。

7.1.3 数据类型

基本数据类型有以下六种：

- 整数型(integer)
- 长整数型(long)
- 单精度数型(single)
- 双精度数型(double)
- 可变长度字符串型(string)
- 固定长度字符串型(string * L)

● 整数型

它是由正号或负号和数字组成的数。在计算机中，用 16 位有符号的二进制补码存放。例如：

+7 的机器码是 0000000000000111，　　−7 的机器码是 1111111111111001

−7 是补码表示的，它是由 +7 的原码“取反加 1”换算来的。

因此，存放一个整数要占用两个字节的存贮空间，数的最小到最大的范围是

−32768～+32767

● 长整数型

存放一个长整数要占用四个字节的存贮空间，数的最小到最大的范围是

−2147483648～+2147483647

● 单精度数型

它是用 IEEE 格式表示的浮点数，单精度数要占用四个字节的存贮空间。

IEEE(Institute of Electrical and Electronics Engineers)格式规定 32 位的单精度数，符号占 1 位，指数占 8 位，尾数占 23 位，另有一个附加的隐含位。采用这种格式的好处是可以使结果更精确，并且可以使用 8087、80287 数学协处理器。单精度数准确到 7 位十进

制数。

负数的范围是 −3.402823E+38～−1.40129E−45

正数的范围是 1.40129E−45～3.402823E+38

● 双精度数型

一个双精度数要占用8个字节的存贮空间。IEEE 格式规定 64 位的双精度数，符号占1位，指数占 11 位，尾数 52 位，另有一个附加的隐含位。

双精度数准确到 15 或 16 位十进制数。

负数的范围是 −1.797693134862316D+308～−4.94065D−324

正数的范围是 4.94065D−324～1.797693134862316D+308

● 可变长度字符串型

可变长度字符串是一个可多达 32767 个的字符序列。字符代码是标准的 ASCII 码 0～127 和扩充的 ASCII 码 128～255，每个字符占 1 个字节的存贮空间。

● 固定长度字符串型

它的长度最大值和字符代码与可变长度字符串相同，不同的是它的长度要经过事先说明，在使用中长度不变。

● 用户自定义的数据类型

用电脑处理较为复杂的数据时，只有基本数据类型是不够的。例如：一份学生成绩单，至少包括学号、姓名、各学科成绩，每个学生都具有相同的这些项目，于是需要一种称为“记录”数据结构的类型。Quick BASIC 允许用户用 TYPE…END TYPE 的语句来定义这样的数据类型，称它为用户自定义类型。

7.1.4 常量与变量

● 数值常量

1. 整数

一般多用带符号的十进制数表示，如 84，−6，+21 等，也可以用八进制数或十六进制数表示。

2. 长整数

一般多用带符号的十进制数表示，如 95000，−400141 等，也可以用八进制或十六进

制数表示。

3. 定点实数

它是带小数点的有符号数。如 9.0846，－0.0001 等。

4. 浮点单精度数

它是用指数形式表示的正数或负数。如 5.935E－7，－234E6，字母 E 的左边尾数部份可以是整数或定点实数，字母 E 的右边指数部份是一个带符号的整数。以下的形式也可以表示单精度数：

以符号"!"结尾的数，如 3247!；小于 15 位的定点实数，数尾不带"#"，如 8.73219，7412.0 等。

5. 浮点双精度数

和单精度相似，不同的只是把表示指数的符号 E 换为 D 就是了。例如 2.367D21，0.12345678D－8 等，以下的形式也可以表示双精度数：以符号"#"结尾的数，如 3247#；多于 15 位的定点实数，数尾没有符号"!"，如 25478123.16724115。

● 字符常量

字符常量，又称为字符串，它是用双引号括起来的一串字符。例如"This is a book"，其中空格也是字符。字符串的长度要把空格计算在内。

● 变量名

每一个变量在电脑中都要占用一定的内存单元，变量的值就存放在那里。根据所存放数据的类型来分，有数值变量和字符串变量；根据所存放数据的结构来分，有简单变量和下标变量。

在程序中，往往需要多个变量，因此，必须用变量名来区别它们。变量名的命名规则是：

1. 变量名由以下字符组成一个字符串：英文字母、数字、小数点、类型说明符；
2. 字符串的字符数最多为 40；
3. 变量名的第一个字符必须是字母；
4. 类型说明符必须放在变量名末尾；
5. Quick BASIC 中的保留关键字不能做变量名。

例：　A，A1，SUM，TAL，String1，CHR $，KEN · TOM 等都是合法的变量名；
LOG，5E，DJS－130，TCA $ B!，X/Y 等都是不合法的变量名。

取名要见名知义：变量名的含义与程序中代表的事物尽量一致，便于阅读理解，比如用 SUM（或 S）代表和，用 Difference（或 D）代表差等。

在见名知义的原则下，尽可能简单明了，力求变量名不要太长，尽管允许一个变量名可以多达 40 个字符，但太长了也不便阅读和书写。

● 变量的类型

在程序中必须正确地选用变量的类型，并且要对变量的类型进行说明。下面介绍说明变量类型的方法：

1. 在变量名后加类型说明符。

表 7—1　类型说明符及用法

说明符	变量类型	例　　子
%	整 型	X%,INT%
&	长整型	T&,Start&
!	单精度型	S!,TOTAL!
#	双精度型	D#,PRODUCT#
$	字符串型	A$,CHR$

【说明】

这五个类型符中，只有单精度型的说明符可以省略。因此，所有不带类型说明符的变量名，都隐含为单精度型。如 S,TOTAL 是单精度变量。

若变量名相同而只有类型说明符不同，表示的是不同的变量，例如：BIG#，BIG%是两个不相同的变量。

2. 用类型定义语句进行说明

类型定义语句的格式：

DEFINT <字母表>[,<字母表>]...　　定义整型
DEFLNG <字母表>[,<字母表>]...　　定义长整型
DEFSNG <字母表>[,<字母表>]...　　定义单精度型
DEFDBL <字母表>[,<字母表>]...　　定义双精度型
DEFSTR <字母表>[,<字母表>]...　　定义字符串型

【说明】

<字母表>　　由用户自己确定的字母范围，例如 A—Z 或单个的字母。
[ ]　　表示可以选择的项
…　　表示可有多个可选项

以整型数的定义语句为例：

DEFINT A　　以 A 开头的变量名均为整型
DEFINT I—N　　以 I,J,K,L,M,N 开头的变量名均为整型
DEFINT B,G,P—T　　以 B,G,P,Q,R,S,T 开头的变量名均为整型
DEFINT —C　　隐含以开始字母为 A，即 A,B,C 开头的变量名为整型

其它类型定义语句的用法与此相同，不一一举例了。

(1)类型说明符(%、&、!、#、$)优先级高于类型定义语句。例如下列程序

```
DEFINT A-Z
ABST=32
ALPHA$ ="ALPHA"
```

该程序中的变量 ABST 是 A 开头的，定义为整型，而 ALPHA 虽然是以 A 开头的，但它有类型说明符，所以它是字符串型，并不遵守类型定义语句的规则。

(2)类型定义语句一般写在模块的开头，对整个模块起作用。

3. 在 DIM、COMMON、SHARED、REDIM、STATIC 等语句中定义变量类型。

7.1.5 标准函数

在程序设计中往往会遇到许多经常处理的问题，如计算乘方、方根，产生一批随机数，把小数进行四舍五入的处理等等。人们把这些使用频率很高的处理问题的步骤编制成程序，并给每一个程序取一个函数名，存入函数库中，这些函数称之为标准函数。它的含义比数学中的函数要广泛得多。标准函数包含了数学中的一些数值处理函数，也包含了许多非数值处理的函数。

● 常用标准函数

QB 中常用标准函数如表 7—2 所示。

表 7—2　QB 中的常用标准函数

函　数	功　　能
SIN(X)	求 X 的正弦值 sinX，X 必须用弧度作单位
COS(X)	求 X 的余弦值 cosX，X 必须用弧度作单位
TAN(X)	求 X 的正切值 tanX，X 必须用弧度作单位
ATN(X)	求 X 的反正切值 arctg X，函数是弧度
ABS(X)	求 X 的绝对值，|X|
EXP(X)	求以 e 为底的指数 X 的乘幂，e=2.7182
LOG(X)	求 X 的自然对数 lnX，X>0
SQR(X)	求 X 的平方根，$\sqrt{X}$，X≥0
INT(X)	取不大于 X 的最大整数
RND(X)	产生一个在(0,1)之间的单精度随机数

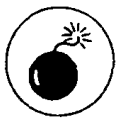
● 标准函数的引用

大多数标准函数包括函数名和自变量两部分，圆括号前是函数名，圆括号内是自变量，如：

SIN　　(X)

函数名　自变量

自变量可以是常量，也可以是变量、表达式，还可以是标准函数，如 SIN(3.14159/2)、SQR(5*X)等。



自变量必须用圆括号括起来。

函数 LOG 是求自然对数，即求以 e 为底的对数 $\log_e X$，如果要求以十为底的对数，应该用换底公式转换：

lgX=lnx/ln10

7.1.6 表达式

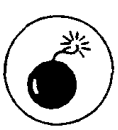
在 QB 中有以下三类运算符和表达式：

- 算术运算符和算术表达式
- 关系运算符和关系表达式
- 逻辑运算符和逻辑表达式

● 算术运算符和算术表达式

1. 算术运算符

+、−、*、/、\、MOD



/和\的区别："/"是浮点除法运算符，"\"是整型除法运算符。

例如：1/2=0.5　1\2=0。

模运算 MOD：

A　MOD　B =A−(A\B)*B

A 为被除数，B 为除数。

例如：9 MOD 7 的值为 2，16 MOD 25 的值为 16。

2. 算术表达式

用算术运算符、圆括号和运算量(常量、变量、函数)连接起来的式子就是 QB 算术表达式。

(1)算术表达式中，每一个符号都占一格，而且一律并排写在一行，不能在上角写方次如 X^2，或在下角写下标如 a_1。

(2)代数式中因子乘积的乘号可以省略，在算术表达式中一律不能省。例如 2SIN(2X+1)必须写成 2*SIN(2*X+1)

(3)凡是希腊字母的符号，如π,α,β,γ等不能出现在算术表达式中。例如，πR^2 中的π，只能用它的值代替。α,β,γ等用变量名代替。

(4)代数式的分式，可以在横线上下写分子分母，写成算术表达式时，不能改变原式的运算顺序，例如$\frac{9-3}{5+2}$应当写成(9−3)/(5+2)。

如果不加括号：9−3/5+2，这就改变了运算顺序，与原式的结果不一样了。

(5)算术运算符不能相邻，如$\frac{X}{-Y}$，应写为 X/(−Y)。

3. 算术运算的优先级别

在算术表达式中包含各种算术运算符，必须规定各个运算的先后顺序，这就是算术运算的优先级别问题。

QB规定各算术运算符的优先级别如下，由高到低排列：

()——→函数——→∧——→*/——→\——→MOD——→+−

● 关系运算符和关系表达式

关系表达式是用关系运算符把两个算术表达式(或两个字符表达式)联接起来的式子。

1. 关系运算符有六种，详见第二章表2.1，此处从略。

2. 关系表达式的一般形式

<算术表达式1>　关系运算符　<算术表达式2>

或<字符表达式1>　关系运算符　<字符表达式2>

3. 关系表达式的运算是对运算符两边的表达式进行比较，其结果是一个逻辑值，只有两种可能的结果：

(1) 两边表达式的关系满足关系运算符的要求，称为条件成立，逻辑值为“真”(用非0的数表示)，在系统中经常用“−1”表示真。

(2) 两边表达式的关系不满足关系运算符的要求，称为条件不成立，逻辑值为“假”(用“0”表示)。

● 逻辑运算符与逻辑表达式

1. 逻辑运算的基本概念

逻辑代数(Logical algebra)又称布尔代数(Boolean algebra)是数学中的一个重要分支，它是计算机的基础理论之一。这里只就几种基本的逻辑运算作一简要介绍。

(1) 逻辑与(AND)：当两个或两个以上的条件都同时满足时，某事物才成立，就称为逻辑与的关系成立。例如，高考录取新生，要求政治条件、健康状况、考试分数都必须达到合格要求才能被录取，即三个条件都为真，录取才为真。

(2) 逻辑或(OR)：当两个或两个以上的条件中只要有一个条件满足时，某事物就成

立。例如某校开干部会，要求“正班长或副班长参加”，也就是说只要其中之一参加就可以了。

(3) 逻辑非(NOT)：表示某一事物两个相反的状态。例如说某一电源开关，接通时为“1”，断开时为“0”，则说断开是接通的非，C=$\overline{A}$。$\overline{A}$表示 A 的非，即 NOT A 的一种表示法。

表 7—3　逻辑真值表

A B	A AND B	A OR B	NOT A	NOT B
0　0	0	0	1	1
0　1	0	1	1	0
1　0	0	1	0	1
1　1	1	1	0	0

2. QB 中的逻辑运算符如表 7—4 所示。

表 7—4　QB 常用逻辑运算符

逻辑运算符	含　义	运算优先级别
NOT	逻辑非	1
AND	逻辑与	2
OR	逻辑或	3

注：运算优先级别以“1”最高，依次一级低一级。

3. 逻辑表达式

它是用逻辑运算符把关系表达式联接起来的式子。

例　X>=0 AND X<100

Y<=0 OR Y>100

X>0 AND X<5 AND Y>0 AND Y<3

由此可见，用逻辑关系运算符可以很方便地表达多个条件的关系运算，如上例中那样表达某变量在某一区间的取值范围。下面的写法是错误的：

100>X>=0　　(错误)

X>=0,X<100　　(错误)

4. 逻辑运算的优先级别

在一个复杂的逻辑表达式中，往往同时含有算术运算符、关系运算符和逻辑运算符，它们在运算时是有一定优先级别的，QB 的规定如下：

首先，计算算术表达式的值；

其次，计算关系表达式的逻辑值；

最后，计算逻辑表达式的逻辑值。

在算术表达式的计算中，优先级别，上面已作介绍。

在关系表达式的计算中，六个关系运算符，按在式子中出现的顺序由左至右进行比较。在逻辑表达式的计算中，优先级别，按表7－4中的规定，从高到低依次为：NOT、AND、OR。

7.2 输入与输出

7.2.1 赋值语句（LET）

【格式】

〔LET〕　〈变量名〉　＝｛〈常量〉|〈变量〉|〈表达式〉｝

其中各部分的含义如下：

LET　　赋值语句的关键字

＝　　赋值号

〔〕　　方括号表示该部分是可选的，〔LET〕表示关键字LET可以写，也可以省略

<>　　尖括号表示是由用户定义的内容

｛｝　　花括号表示在该部分内容中选择一个，各项之间用“|”分隔

【功能】

把赋值号右边表达式的值赋给左边的变量。在计算机中，变量名代表一个存贮单元，给变量赋值，就是把一个表达式的值，存入到这个存贮单元中去。它主要用于数据运算，但也可用于简单的数据输入。

【使用说明】

1. 赋值号右边的数据类型与左边的变量类型必须一致。

2. 如果赋值号右边是一个表达式，执行时先进行右边表达式的计算，然后把得到的值再赋值给左边的变量。即先计算，后赋值。

3. 赋值号“＝”与数学运算中的等号意义不同，不可混淆。

7.2.2 变量数值交换语句(SWAP)

【格式】

SWAP　〈变量名1〉，〈变量名2〉

【功能】

交换两个类型相同的变量值。

例：

```
'Example QB7—0.bas
A = 5 : B = 8        '变量 A，B 赋值
PRINT A , B
SWAP A , B           '变量 A，B 交换数值
PRINT A , B
END
```

7.2.3 屏幕显示输出语句(PRINT)

【格式】

PRINT〔〈输出项 1〉〕〔{ ,|;}〈输出项 2〉〕… { ,|;}

其中：

" ,"　　输出项之间的分隔符，表示按标准格式输出。

" ;"　　输出项之间的分隔符，表示按紧凑格式输出。

＜输出项＞　可以是常量、变量、表达式。

"…"　　表示可重复多个。

【功能】

把输出项的值显示到屏幕上。输出项是常量、变量、表达式时，输出它们的值；输出项是用双引号括起来的字符串时，原样输出字符串；缺省输出项时，输出空行。

【使用说明】

1. 一个 PRINT 语句可以输出一个或多个输出项，当多项输出时，中间用分隔符","或";"分隔。

例如：　PRINT 5 ,9 ,21

　　PRINT A ;B ;A+B

2. PRINT 语句用以原样照印字符串时，字符串要用双引号括起。

例如：PRINT "GOOD!"

3. 当输出项为表达式时，PRINT 语句具有计算功能，即先计算后输出。

4. 为了使输出结果清晰，人们对输出格式常有不同要求，输出格式可以用输出项之间的分隔符来指定，所用分隔符有逗号","和分号";"两种；还可用专门的函数来指定，专用函数有输出定位函数 TAB 和输出空格函数 SPC 。

● 标准格式输出

当 PRINT 语句输出项之间用逗号分隔时，称为标准格式输出。QB 把一行分成 5 个显示区域，每个显示区域的宽度是 14 个字符。

例：

【源程序】

```
'Example QB7－1.BAS
PRINT  "1","2","3","4","5"
PRINT  10，31，－8，12，－23
PRINT  1，2，3，4，5，6，7，8
END
```

【运行结果】

```
1             2             3             4             5
 10            31           －8            12           －23
 1             2             3             4             5
 6             7             8
```

● 紧凑格式输出

当PRINT语句输出项之间用分号分隔时，输出数据一个紧挨一个，称为紧凑格式输出。紧凑格式输出规则如下：如果输出的是数值量，每个数的后边有一空格，每个数的前边有一个符号位，当数为正数时，“＋”号可以省略但仍保留一个空格；如果输出的是字符串，则串的前后不留空格。

例：

【源程序】

```
'Examlpe QB7－2.BAS
PRINT "123456789A123456789B"
PRINT 8；0；－10；18；－12.6
PRINT "Q－"；"BASIC"
END
```

【运行结果】

```
123456789A123456789B
 8  0 -10  18 -12.6
Q-BASIC
```

● 输出定位函数TAB

【格式】

TAB（＜算术表达式＞）

其中：

表达式的值必须是大于或等于零的整数(如果是小数,则按四舍五入取整),该整数即为指定输出位置的列号。

例:

【源程序】

```
'Example QB7-3.BAS
PRINT "123456789A123456789B123456789C"
A = -23 : B = 5
PRINT TAB(3) ; "A+B=" ; TAB(8) ; A ; TAB(12) ; "+" ;
PRINT TAB(14) ; B ; TAB(17) ; "=" ; TAB(20) ; A+B
END
```

【运行结果】

```
123456789A123456789B123456789C
  A+B= -23 +  5 =  -18
```

【使用规则】

1. 输出项跟在 TAB 函数后面,用分号";"分隔,输出项按 TAB 函数定位的列号输出。

2. TAB 函数定位的列号超过行宽时,则在下一行按行宽的余数输出。例如行宽为 80 列,当 TAB 定位于 90 列时,则实际定位下一行的第 10 列。

3. TAB 函数后若没有输出项,则输出无效。但有后续 PRINT 语句时,可作后续 PRINT 语句的定位点。

4. 在一个 PRINT 语句中,可以包含多个 TAB 函数,各个函数之间用分号";"分隔。

● 输出空格函数 SPC

【格式】

SPC (<算术表达式>)

其中表达式的值必须是大于或等于零的整数(如果是小数,则按四舍五入取整),该整数即为跳过的空格数。

例:用 SPC 函数打印一个表框

【源程序】

```
'Example QB7-4.BAS
PRINT "123456789A123456789B123456789C123456789D"
PRINT
PRINT SPC(12) ; "**************"
PRINT SPC(12) ;"*" ; SPC(12) ; " *"
PRINT SPC(12) ; "*" ; SPC(4) ; "LIST" ; SPC(4) ; "*"
PRINT SPC(12) ; "*" ; SPC(12) ; "*"
```

```
PRINT SPC(12) ; "**************"
END
```

【运行结果】

```
123456789A123456789B123456789C123456789D

            **************
            *            *
            *    LIST    *
            *            *
            **************
```

● 自选格式输出

【格式】

PRINT USING "〈格式字符串表达式〉";〈输出项 1〉[,〈输出项 2〉] …[{, |;}]

其中：

＜格式字符串表达式＞——用于定义输出项的输出格式。

【功能】

按格式字符串表达式定义的输出格式输出字串或数值。

【说明】

1. 一个 PRINT USING 语句中只能有一个格式字符串表达式。

2. 表达式由格式代码及其它字符组成，表达式要用双引号括起来。

格式代码共有 14 种，它们是：

用于输出字符串的 3 种：!、\\ 、&。

用于输出数值的 11 种：#、^ 、+、-、$ $、* *、* * $、.、,、%、-。

文字字符可以是：0—9 数字；大小写英文字母；某些键盘符等。

"#"、"."格式码的输出：

(1) 输出数字的长度及小数部分的位数完全由格式代码"#"号确定。

"# # # # # #"：表示输出 6 位整数，长度为 6 列。

"# # # #. #"：表示整数部分 4 位，小数部分 1 位，小数点占 1 位，总长度仍是 6 位。

"# # #. # #"：表示整数部分 3 位，小数部分 2 位，小数点占 1 位，总长度仍是 6 位。

(2) 数值指定的位数不得超过 24，否则将出现 Illegal function call 的出错信息。

(3) 所有数字都是向右对齐，定义多余的整数部分补空格，多余的小数部分补 0，若小数部分定义不足，则尾数自动四舍五入。

(4) 正数不显示符号，负数显示"-"号。

例：

【源程序】

```
'Example QB7—5.BAS
A = 126 : B = 21.2 : C = -2.638 : D = 0.273
```

```
PRINT "123456789A123456789B123456789C"
PRINT USING " ######## " ; A ; B ; C ; D
PRINT USING " ####. # " ; A ; B ; C ; D
PRINT USING " ###. ## " ; A ; B ; C ; D
PRINT USING " ###.## " ; B
PRINT USING " ###.## " ; C
PRINT USING " ###.## " ; D
END
```

【运行结果】

```
123456789A123456789B123456789C
   126     21    -3      0
 126.0   21.2  -2.6   0.3
126.00  21.20 -2.64  0.27
 21.20
 -2.64
  0.27
```

7.2.4 键盘输入语句(INPUT)

【格式】

INPUT 〔;〕〔"〈提示字符串〉" {,|;}〕〈变量名 1〉〔,〈变量名 2〉〕…

其中：

〔;〕　任选。有此分号时，当用户输入完数据并回车后，光标仍停留在同一行上。没有此分号，则换行。

＜提示字符串＞　可选。用于说明输入的内容，必须用双引号括起来。

{;|,}　任选其一，当选择分号时，在显示出提示字符串之后，跟有"?"；当选择逗号时，提示字符串之后没有"?"。

【功能】

程序执行到此语句时将暂停执行，屏幕显示"?"(如果语句中有提示字符串，将原样显示字符串)，表示向用户索要准备给变量赋值的数据。这时用户从键盘上输入数据并回车后，变量则被赋值。

【使用说明】

该语句的特点是在程序运行中，通过键盘给变量赋值。输入数据的规则是：

1. 对于一个包含有多个变量的 INPUT 语句，必须在一行将所有数据按变量的个数和排列的顺序依次输入，并用逗号分隔。数据个数多或少于变量的个数都将产生错误信息：

Redo from start ，此时用户应重新输入数据。

2. 输入的数据必须与变量的类型一致。

3. 从键盘输入字符串时，不要求用双引号括起来。但是，当字符串本身包含有逗号或空格时，必须用双引号括起来。

4. 最后一个数据项输入完之后必须回车。

例：任意输入某一同学的数学、物理、化学三门课的考试成绩，计算平均分

【源程序】

```
'Example QB7—6.bas
INPUT "Mathematics:", A
INPUT "Physics:", B
INPUT "Chemistry:", C
T = (A + B + C) / 3
PRINT "Average:"; SPC(3);
PRINT USING "###.##"; T
END
```

【运行结果】

```
Mathematics:   78
Physics:       90
Chemistry:     86
Average:       84.67
```

7.2.5 读数、置数语句(READ/DATA)

【格式】

读数语句：READ〈变量名 1〉[,〈变量名 2〉]…

置数语句：DATA〈数据项 1〉[,〈数据项 2〉]…

【功能】

置数语句把数据预先存放在内存中的一块数据区内，并把读数指针指向第一个数据，执行读数语句时，第一个数据送给读数语句的第一个变量，读数指针移向下一个数据，依次读入以下各个变量。

例：

【源程序】

```
' Example QB7—7.BAS
READ    A , B , C , D
READ    E , F , G , H
PRINT   A , B , C , D , E , F , G , H
READ    X $ , X , Y $ , Y , Z $ , Z
PRINT   X $ ; X , Y $ ; Y , Z $ ; Z
```

```
DATA    -1 , 8 , 0 , 6 , 4
DATA    3 , -5,2
DATA    X= , 8.26 , , , Z= , -4.3
END
```

【运行结果】

```
-1              8           0           6           4
3               -5          2
x= 8.26         0           Z=-4.3
```

【使用说明】

1. READ 与 DATA 语句是配合使用的。READ 语句是执行语句;DATA 语句是非执行语句,放在程序中任何位置都可以,习惯上集中放在程序的最后,END 语句之前。

2. 有多个 DATA 语句时,计算机按 DATA 语句出现的先后顺序,以及在每个 DATA 语句中数据排列的顺序,依次把每一个数据置于内存储器的一段连续的数据区中。

3. 在一个程序中 DATA 语句常量的总个数如果少于 READ 语句的变量的总个数,执行时会产生一个出错信息:Out of Data(即缺少数据),并停止程序执行;如果常数总个数多于变量总个数,不会出错,只是多余的常量没有被读取。

4. 由于所有 DATA 语句中数据的存放是连续的,所以各个 READ 语句的变量相互之间也都是连续的。因此,数据与被赋值变量的位置顺序与类型要严格对应,但 DATA 语句与 READ 语句的个数不必一一对应。如第 2 行的 READ 语句中有四个变量,而第 7 行的 DATA 语句中有五个数据。

5. DATA 语句中的数据只能是常量,不允许出现变量、表达式或函数;DATA 语句中的字符串常量可以不加双引号,但是,当字符串中包含有逗号、冒号、后缀空格等一些特定字符时,应加双引号;DATA 语句中允许有空的数据项,如果空项被读入到数值变量中,则变量值为 0,如果空项被读入到字符串变量中,则变量为空串。

6. 在 QB 中,DATA 语句只能放在模块级代码中。

7.2.6 符号常数说明语句 (CONST)

【格式】

CONST〈符号常数名 1〉=〈表达式 1〉[,〈符号常数名 2〉 =〈表达式 2〉] …

其中:

〈符号常数名〉 遵守变量名的规则。也可以在名字后附加类型说明符(%、&、!、# 或 $),用以说明它的类型。

〈表达式〉 由数值常数、字符串常数、算术运算符或逻辑运算符组成,其中不能有变量或函数。

【功能】

定义符号常数代替的数值或字符串值。

【使用说明】

1. 常数的类型由类型说明符决定，如果类型说明符省略，则常数类型由表达式决定。

例如：CONST MIN%=10，MAX =MIN+100

其中，符号常数 MIN 用类型说明符%说明其为整型；符号常数 MAX 省略了类型说明符，但对应的表达式的值为整型数，因此其类型由表达式确定为整型。

2. CONST 语句一般放在程序的开始，因为符号常数必须先定义，后引用。

3. 符号常数是一个固定数值的符号，不应当作变量用。一经定义，不能再变。重新定义或作为变量赋值都是不允许的。

4. 在子程序中定义的符号常数是局部的，只在子程序中起作用；在过程外面说明的符号常数，则在整个模块起作用。

例：我国现有人口为 12 亿，按 0.005 的增长率增加，100 年后，人口总数是多少？

【算法】

人口增长的规律按公式 $S=A(1+X)^n$ 计算

设 A=12，X=0.005，N=100

【源程序】

```
'Example QB7-8.BAS
A=12 : X=0.005 : N=100
S=A * (1+X) ^ 100
PRINT "S=";S
END
```

【运行结果】

```
S=19.76002
```

7.3 选择结构语句

7.3.1 块 IF 语句

【语句格式】

IF <条件> THEN

…} S1<语句组 1>

〔ELSE〕

…} S2<语句组 2>

END IF

其中：

<条件>	是一个关系表达式或逻辑表达式
S1	条件成立时执行的语句组
S2	条件不成立时执行的语句组

【功能】

如果 IF 语句中的条件成立的话，就执行 S1 的语句；S1 执行后，程序跳过 S2，转到 END IF 语句的下一个语句去继续执行；

如果 IF 语句中的条件不成立的话，就跳过 S1，执行 ELSE 后的 S2 的语句，执行完 S2 后，转到 END IF 的下一个语句去继续执行。总之，它对两个分支，只取其一执行。

例：任意输入三个数 A、B、C，找出其中最大的数和最小的数

【算法】

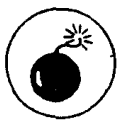
1. 先比较 A 与 B，大者赋值给 MAX，小者赋值给 MIN；
2. 再将 MAX 与 C 比较，大者存入 MAX；
3. 再将 MIN 与 C 比较，小者存入 MIN；
4. 输出 MAX，MIN。

【源程序】

```
'Example QB7－9.bas
INPUT "A,B,C=";A,B,C
IF A>B THEN
  MAX=A
  MIN=B
ELSE
  MAX=B
  MIN=A
END IF
IF C>MAX THEN
  MAX=C
END IF
IF C>MIN THEN
ELSE
  MIN=C
END IF
PRINT "MAX=";MAX;"MIN=";MIN
END
```

【程序说明】

块 IF 中的 S1 和 S2 可以同时存在，这是它的基本形式，但它可以缺省 S1，或缺省 S2，这就是它的特殊形式。该程序中包含了三个 IF 块：第一个是完整的 IF 块；第二个缺省 S2，ELSE 可省去不写；第三个缺省 S1，ELSE 语句不能省。



每个块 IF 必须以 END IF 结束，不可缺省，而且 END IF 必须与 IF THEN 配对。

7.3.2　行 IF 语句

【格式】

IF ＜条件＞ THEN {＜语句组 1＞[ELSE{＜语句组 2＞}]}

其中：

＜条件＞　　是一个关系表达式或逻辑表达式

语句组 1　　条件成立时，执行该语句组

语句组 2　　条件不成立时，执行该语句组

语句组 1、语句组 2 中包含多个语句时，各语句要用“:”分隔。

【功能】

和块 IF 语句相同，但当语句组有多个语句时，写在一行里就显得不清晰，因此只在简单分支时使用。一般使用块 IF 语句更灵活，更能体现程序的结构化。

例：从键盘上输入三个数，找出其中最大的数

【算法】

1. 输入 A、B、C 三个数；
2. A 存入一变量 BIG；
3. 如果 B＞BIG，把 B 存入 BIG；
4. 如果 C＞BIG，把 C 存入 BIG；
5. 打印 BIG 的值

【源程序】

```
'Example QB7－10.bas
INPUT A,B,C
BIG=A
IF B>BIG THEN BIG=B
IF C>BIG THEN BIG=C
PRINT "The bigest is";BIG
END
```

例:把任意三个数按由大到小排列打印,用嵌套的块 IF 写程序

【源程序】

```
'Example QB7-11.bas
INPUT A,B,C
IF A>B THEN
ELSE
  SWAP A,B
ENDIF
IF B>C THEN
ELSE
  SWAP B,C
  IF A>B THEN
  ELSE
  SWAP A,B
  ENDIF
ENDIF
PRINT A,B,C
END
```

【程序说明】

在块 IF 结构中的两个语句块(S1、S2)中均可再包括块 IF 结构,这就称为嵌套。嵌套时无论是在 S1 块或在 S2 块中块 IF 语句必须是一个完整的块 IF 构。不能跨在 S1 和 S2 两块之间。在嵌套中的块 IF 语句,又可以再次嵌套,即允许多层嵌套。

7.3.3 多分支的块 IF 语句

在块 IF 的结构中,增加一个 ELSEIF 语句,就扩充成为多分支的块 IF 结构。

【格式】

```
IF <条件 1> THEN

    …}S1(块 1)

[ELSEIF <条件 2> THEN]
    … }S2(块 2)
[ELSEIF <条件 3> THEN]
    …}S3(块 3)                    可以包括多个 ELSEIF 语句和相应的语句块
    …
[ELSEIF <条件 n> THEN]
    …}Sn(块 n)
[ELSE]

    …}Sn+1(ELSE 块)
```

ENDIF

【功能】

当条件 1 的逻辑值为真时，就执行语句块 1(即 S1)，执行完后就退出 S1，转到 ENDIF 语句的下一个语句继续执行；如果条件 1 的逻辑值为假时，就跳过 S1，执行 ELSEIF <条件 2> THEN 语句，当条件 2 的逻辑值为真时，就执行 S2，执行完后，退出 S2，转到 ENDIF 语句的下一个语句；如果条件 2 的逻辑值为假时，就又跳过 S2，执行下一个 ELSEIF 语句。依此类推，有多少个 ELSEIF 语句，就有多少个分支，选择其一执行相应的语句块，然后退出该语句块。

如果所有条件都不满足，则执行 ELSE 后的语句，然后退出该语句块。

例：计算个人所得税，假设按以下规定征收：以月收入计征，单位为元

表 7—2　个人所得税征收率

收入金额	征税率
500 以下	免征
500～1000	5%(扣除免征部分)
1000～5000	10%(扣除免征部分)
5000～10000	20%(扣除免征部分)
10000～20000	30%(扣除免征部分)
20000 以上	45%(扣除免征部分)

【算法】

六种不同的收税标准构成六个条件的多路选择，用多分支的块 IF 就可写出程序。

【源程序】

```
'Example QB7—12. bas
'Income tax computation
INPUT "Get the income : ";Income
IF Income<500 THEN
  PRINT "Tax Free"
ELSEIF Income<1000 THEN
  Tax=(Income—500) * 0. 05
ELSEIF Income<5000 THEN
  Tax=(Income—500) * 0. 10
ELSEIF Income<10000 THEN
  Tax=(Income—500) * 0. 20
ELSEIF Income<20000 THEN
```

```
    Tax=(Income-500) * 0.30
ELSE
    Tax=(Income-500) * 0.45
END IF
    PRINT "TAX=";Tax
END
```

7.3.4 SELECT CASE 语句

这个语句的格式、功能、用法已在前面介绍过了，此处从略.

7.4 循环结构语句

7.4.1 计数循环语句(FOR—NEXT)

【语句格式】

FOR <循环变量>=<初值> TO <终值> [STEP <步长>]
　　<循环体>
NEXT <循环变量>

其中：

<循环变量>又称循环控制变量，是简单变量。

<初值>，<终值>，<步长>可以是整型或实型的常量、变量或算术表达式。当步长为 1 时，[STEP <步长>]可以省略。

我们以下面程序中的 FOR 循环为例来说明 FOR—NEXT 循环的执行过程。

```
FOR I=1 TO T        'I 是循环变量,1 是初值,T 是终值,步长值为 1,省略了
M=M * I             ' 这是要重复执行的循环体
NEXT I              ' 循环终端语句,标志循环结束
```

【执行过程】

1. 首先，把初值 1 放到循环变量 I 中，即 I=1；

2. 初值和终值比较，假定终值 T 为 5，如果初值没有超过终值，就执行循环体内的语句；如果初值已经超过终值，就不执行循环体内的语句，即不执行以下步骤 3、4、5，而是转到步骤 6，退出循环，执行 NEXT 之后的语句；

3. 初值没有超过终值，执行循环体内的语句：M 和 I 相乘再放回到 M 变量中，比如第一次执行循环体时，M=1，I=1，其乘积为 1，放回 M，M=1，以后，每执行一次循环体，M 的值都在改变；

4. 执行完循环体内的语句之后，将循环变量加步长值，再送回循环变量，即循环变量按步长增值，比如步长为 1，I=I+1，所以，第一次执行循环体后，循环变量 I 就增值为 2，

以后随着循环次数的增加，I 不断变化，直到超过终值；

5. 循环变量增值后，执行转到步骤 2，进行下一个循环；

6. 初值已经超过终值，退出循环，执行 NEXT 之后的语句。

【说明】

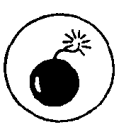
1. FOR 语句称为循环说明语句，也是循环的入口。执行该语句时循环变量的取值从初值开始，每循环一次增加一个步长，直到超过终值为止，由此决定循环的执行次数。循环执行的次数可以用下面的公式计算出来。

循环次数＝INT((终值－初值) /步长)＋1

凡是计算出的循环次数小于等于 0 的，循环次数为 0。

2. NEXT 语句称为循环结束语句，是循环的出口。执行该语句时，循环变量取下一个值，即把控制循环变量的当前值增加一个步长。

3. FOR 和 NEXT 之间的语句称为循环体，它是循环语句的主体，是需要多次重复执行的部分，可以由一个或多个语句组成。



FOR 语句和 NEXT 语句必须配对，而且同一循环中 FOR 语句和 NEXT 语句中的循环变量必须同名，否则就会出错。

4. 步长值可以是整数，也可以是小数，可以是正值，也可以是负值。

·当步长值为正时，在循环的执行过程中，循环变量按步长递增，一直到循环变量的值变化到大于终值时，循环停止。例如：

```
FOR I=1 TO 10 STEP 2.5
```

步长值为正时，终止循环的条件是终值必须大于初值。

·如果步长值为负，在执行循环的过程中，循环变量按步长递减，当循环变量值变化到小于终值时，循环才会停止。例如：

```
FOR X=5 TO 1 STEP -1
```

步长值为负时，终止循环的条件是终值必须小于初值。

·当步长值为小数时：

```
FOR X=1 TO 3+0.5/2 STEP 0.5
```

当步长值为小数时，实数计算会产生一定误差，为确保循环的预定次数，建议终值加上半个步长。

例：打印－π 到 π 之间的正弦曲线，每隔 10°打一“＊”号

【源程序】

```
'Example QB 7-13.bas
PI=3.14159
FOR X=-PI TO PI+(PI/18/2) STEP PI/18
PRINT INT(X*180/PI);
PRINT TAB(32+20*SIN(X));"*"
NEXT X
```

```
END
```

● 循环的嵌套

循环语句的循环体中如果又包含循环语句，称为循环嵌套，在外层的叫外循环，在内层的叫内循环。在一个循环体内包含一个循环语句的为双重循环。内循环体又可包含循环语句，构成多重循环。多重循环增强了计算处理的能力，能解决一些比较复杂的问题。

双重循环的结构为：

```
FOR<循环变量 1>=<初值 1> TO <终值 1> [STEP <步长 1>]
  FOR<循环变量 2>=<初值 2> TO <终值 2> [STEP <步长 2>]
    <循环体>
  NEXT <循环变量 2>
NEXT <循环变量 1>
```

例：打印九九乘法表

【源程序】

```
'Example QB7—14.bas
    FOR A=1 TO 9
      FOR B=1 TO A
        PRINT USING "#";A;
        PRINT "*";
        PRINT USING "#_=##";B,A*B;
        PRINT " ";
      NEXT B
      PRINT
    NEXT A
    END
```

我国古代数学家张邱建在《算经》里提出了一个不定方程问题。即“百鸡问题”：公鸡每只值五元，母鸡每只值三元，小鸡三只值一元，一百元钱买一百只鸡。问公鸡、母鸡、小鸡各可买多少？

【算法】

设公鸡、母鸡、小鸡各为 x、y、z，可得方程：

$$5x+3y+\frac{z}{3}=100$$

$$x+y+z=100$$

这个不定方程式有多个答案，我们用穷举法求解。

所谓穷举法，就是对所有可能的答案一一列举，从中判断哪些答案是符合题设条件的。

我们分析一下 X、Y、Z 的可能取值范围：公鸡最少 1 只，最多只能买 20 只，故 1≤X<20；母鸡 1≤Y<33；只要 X、Y 值一确定，Z=100−X−Y。在各种 X、Y 值的情况下，一一列举出 X、Y 所能取值的情况。因此用双层循环编写程序。外层循环控制 X(即 FOR X=1 TO 19)，内层循环控制 Y(即 FOR Y=1 TO 33)。

【源程序】

```
'Example QB7−15.bas
PRINT "Cock","Hen","Chiken"
FOR X=1 TO 19
  FOR Y=1 TO 33
    Z=100−X−Y                                    '满足百鸡条件
    IF 5*X+3*Y+Z*3=100 THEN PRINT X,Y,Z          '满足百鸡条件
  NEXT Y
NEXT X
END
```

7.4.2　DO−LOOP 条件循环语句

DO 循环可分为“当型循环”与“直到型循环”两种，而在当型和直到型中又可分为先进行条件判断再执行循环体和先执行循环体再进行条件判断两种类型。由此构成四种格式如下：

类　型	WHILE(当型)	UNTIL (直到型)
先判断后执行	DO WHILE−LOOP	DO UNTIL−LOOP
先执行后判断	DO−LOOP WHILE	DO−LOOP UNTIL

【格式一】

```
DO WHILE <条件>
  <循环体>
LOOP
```

说明：当条件成立时，执行循环体；当条件不成立时，不执行循环体。

【格式二】

```
DO
  <循环体>
LOOP WHILE <条件>
```

说明：首先执行循环体，再判断条件是否成立。如果条件成立，再执行循环体；如果条件不成立，就退出循环体。

【格式三】

```
DO UNTIL <条件>
```

<循环体>
LOOP

说明：当条件不成立时，执行循环体，直到条件成立时，退出循环体。

【格式四】

DO
<循环体>
LOOP UNTIL <条件>

说明：首先执行循环体，再判断条件是否成立。如果条件不成立，再执行循环体；如果条件成立，就退出循环体。

上述四种格式中的<条件>为关系表达式或逻辑表达式。

例：求自然数的平方和S，总和S是大于N的最小数

【源程序一】

```
'Example QB7-16A.bas
INPUT N
  I=0
  S=0
DO WHILE S<=N
  I=I+1
  S=S+I*I
LOOP
  PRINT "S=";S
END
```

【源程序二】

```
'Example QB7-16B.bas
INPUT N
  I=0
  S=0
DO
  I=I+1
  S=S+I*I
LOOP WHILE S<=N
  PRINT "S=";S
END
```

【源程序三】

```
'Example QB7-16C.bas
INPUT N
  I=0
  S=0
DO UNTIL S>N
  I=I+1
  S=S+I*I
LOOP
  PRINT "S=";S
END
```

【源程序四】

```
'Example QB7-16D.bas
INPUT N
  I=0
  S=0
DO
  I=I+1
  S=S+I*I
LOOP UNTIL S>N
  PRINT "S=";S
END
```

【使用说明】

1. DO 循环的格式二、三在程序设计中可以互换，但条件要相反，功能是相同的。例如 7.17B 和 7.17C。

2. DO 循环的嵌套和 FOR 循环相同。

3. 如果循环体内没有控制循环条件的变量，将出现"死循环"。要停止死循环只有通过人工干预的办法，方法是同时按下 Ctrl＋Beark 键来打断循环。程序设计中，应避免这种情况发生。

4. 从循环中途退出的方法

(1) 用 EXIT DO 语句从 DO-LOOP 中退出。

在执行 DO 循环时，除了根据循环条件的成立或不成立而退出循环外，可以用控制语句 EXIT DO 中途跳出循环。EXIT DO 为循环的出口语句。当程序执行到 EXIT DO 语句时，立即转到 LOOP 语句的后续语句继续执行。EXIT DO 语句一般要和 IF 语句一起联用。

(2) 用 EXIT FOR 语句从 FOR-NEXT 中退出

在 FOR 循环中，除正常的通过控制循环次数来终止循环外，也可以用控制语句 EXIT FOR 在 FOR 循环完成之前跳出循环，而执行 NEXT 语句的后续语句。

例：求两个正整数 M、N 的最大公约数

【算法】

用欧几里得(Euclidean)的辗转相除法求解。

【源程序】

```
'Example QB7－17.bas
INPUT M,N
DO WHILE M>0 AND N>0
  R=M MOD N
  DO UNTIL R=0
    M=N
    N=R
    R=M MOD N
  LOOP
  PRINT "The greatest common divisor is";N
  EXIT DO
LOOP
END
```

例：猜数游戏。由随机数发生器产生 100 以内的整数，用户从键盘上猜一个数，计算机会提示大了还是小了，直到猜准为止

【源程序】

```
'Example 7－18.bas
'Number quessing game
```

```
RANDOMIZE TIMER
Number = INT(100 * RND + 1)
Count = 1
DO
  PRINT "Your quess";
  INPUT Quess
  Count = Count + 1
  IF Quess > Number THEN
    PRINT "Too big"
  ELSEIF Quess < Number THEN
    PRINT "Too small"
  ELSE
    PRINT "That's right!"
    PRINT "You needed"; Count; "Quesses"
  END IF
LOOP UNTIL Quess = Number
END
```

7.5 数组

7.5.1 数组和下标变量

● 一维数组

一批相同类型的数的有序集合称为数组。在数学中，一个一维向量 A 表示为

$A(a_1,a_2,a_3,\cdots,a_n)$

这也是一个数组。A 是这个数组的名字，组内的每个成员，分别用下标表示，下标不同，表示的成员也不同，每个成员就称为数组元素，又叫做下标变量。在这个数组中只用一个下标就能确定其中每一个成员，这样的数组元素称为单下标变量，这种数组称为一维数组。

● 二维数组

在数学中的矩阵表示为

$$B=\begin{vmatrix} b11 & b12 & b13 & \cdots & b1n \\ b12 & b22 & b23 & \cdots & b2n \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ bn1 & bn2 & bn3 & \cdots & bnn \end{vmatrix}$$

这也是数组，B是数组名，组内每一个数组元素，要用两个下标才能确定，即要指出它所在的行和列的位置，这样的数组元素称为双下标变量，这种数组称为二维数组。常见的表格就是二维数组，例如，一张学生成绩表：

学生成绩表

学生姓名	政治	语文	外语	数学
张 三	90	85	87	95
李 四	85	90	95	70
王 五	87	70	75	80

这就是一个有三行四列的矩阵，构成为一个二维数组。

下标变量中下标的个数称为数组的维。因此，一个下标的下标变量，构成一维数组，两个下标的下标变量，构成二维数组，三个下标的下标变量，构成为三维数组，有多少个下标的下标变量，就构成多少维的数组，常常又把二维以上的数组称之为多维数组。

总之，数组是相同类型数据的有序集合。它具有两个特点：①数据类型必须相同，不能把不同类型的数据放在一个数组里；②数组内的每一个数据都有确定的位置，按一定顺序排列，而不是互不相关的。

7.5.2 数组和下标变量的表示方法

● 数组的命名

为了区分各个不同的数组，必须给每个数组取一个名字。命名规则与简单变量名的命名规定相同。最好按数组的实际意义取名，见名知义，便于阅读。

● 下标变量的表示

在数学中用右下角的下标字符来表示，如 A_1，在电脑里就无法表示出下角标号，也不能用A1表示，这和简单变量名混淆了。所以QB规定把下标放在数组名后的一对括号中，如用A(1)来表示 A_1，因此，一个下标变量可用数组名后跟一个说明下标的括弧来表示。

多维数组的下标变量有多个下标，各下标之间，要用逗号隔开，例如前面的学生成绩表构成一个二维数组，取名为 score，它的每一个下标变量可以表示如下：

score(1,1)=90, score(1,2)=85, score(1,3)=87, score(1,4)=95

score(2,1)=85, score(2,2)=90, score(2,3)=95, score(2,4)=70

score(3,1)=87, score(3,2)=70, score(3,3)=75, score(3,4)=80

在二维数组的下标变量中，第一个下标表示所在的行，第二个下标表示所在的列。

下标可以用常量、变量、表达式表示。例如：

TEP(I)　　SCORE(J,K)

TEP(M+1)　SCORE(N+1,K+2)

如果下标值是小数，系统将按四舍五入转换为整数。

● 数组类型的说明

在数组名的最末一个字符用以下特定字符表示该数组的类型。

符 号	数组类型	例　子
%	整数型	A%(I),NUM%(J,K)
!	单精度数型	TEP! (I),SCORE! (J,K)
#	双精度数型	B#(I),NUM#(J,K)
$	字符型	N$(I),CHAR$(J,K)

这四个符号中，唯一只有单精度型的类型符“!”可以省略，例如 TEP! (I) 和 TEP(I)是相同类型的。所以，凡是没有类型说明符的，其类型就被隐含说明为单精度型。

7.5.3 DIM 语句

【格式】

DIM 〔SHARED〕<变量名>〔(数组维数的说明)〕〔AS <类型>〕〔,<变量名>〔(数组维数的说明)〕〔AS <类型>〕〕…

【说明】

SHARED　　是一个可选项，如选此项，则模块中所有过程共享数组和简单变量。

<变量名>　为简单变量或数组的名字。

AS <类型>　说明类型，AS 是关键字，<类型>可以是：

INTEGER

LONG

SINGLE

DOUBLE

STRING

自定义类型.

<数组维数的说明> 维数说明的格式如下：

〔下界 TO〕上界〔,〔下界 TO〕上界〕…

关键字 TO 省略时，隐含下界为 0，下标范围只能为等于大于 0 的整数，使用关键字 TO，可以指定下界的范围扩展为－32768～32767。当下标值是小数时，系统将按四舍五入取整。

【功能】

1. 定义简单变量的类型

在 7.1 节中介绍了变量类型说明的三种方法，现介绍用 DIM 说明简单变量类型，其方法如下：

DIM AVERAGE AS INTEGER, PAGE AS STRING * 2

经过 DIM 语句说明，在程序中变量 AVERAGE 就是整型变量。而 PAGE 则是可变字符串变量，如果未作类型说明时，这两个变量却都是隐含的单精度型，这一点读程序时要多留心。DIM 语句只能对一个一个的变量名进行说明，不能象 DEF 类型定义语句那样定义一批变量的类型。

2. 定义数组的维数和类型

例如：

DIM A(20),B(－2 TO 5,10 TO 30)

它说明数组 A 是有 21 个(下标从 0～20)的一维数组，数组 B 是二维数组，且行的下标从－2～5 共 8 行，列的下标是 10～30，共 21 列。

数组 A 和数组 B 若未作任何类型说明，隐含为单精度型。也可用 AS 定义数组类型，例如：DIM B(15) AS INTEGER

【使用说明】

1. QB 允许数组下标的值从－32768 到 32767，范围是宽广的，但必须遵守下界值小于上界值规则。例如：X(10 TO 1)是错误的。

2. 省略关键字 TO，下界隐含为 0，如：X(12)与 X(0 TO 12)等效。

为了改变下标下界隐含值，可在 DIM 语句前加一个下界选择语句 OPTION BASE。

例如：

```
OPTION BASE 1
DIM S(10,15)
```

数组 S 的行下标是 1－10，列下标是 1－15

OPTION BASE 的格式：

```
OPTION BASE n
```

其中 n 只能是 1 或 0，不能是其它数。

它的功能仅仅用以选择下标的最小值。

3. 在程序中使用了未经 DIM 定义的数组，系统隐含其每维的最大下标值为 10，超出这个值时，会出现错误信息：

Subscript out of range。

4. DIM 语句定义的数组维数最大可达 60。

5. 经过 DIM 定义的数值数组的元素全部初始化为 0，字符数组的元素初始化为空串。

6. DIM 是非执行语句，它的作用是由系统分配一片连续的存储区，相当于预定"房间"。因此，程序中必须遵守"先定义后引用"的原则，一般均把 DIM 语句放在程序的开头。

7. DIM 语句中定义数组下标的值，可以是常量，也可以是变量或表达式，如果含有变量必须在 DIM 语句前给变量赋值。例如：

```
INPUT M,N
DIM X(M+1,N+1) AS INTEGER
```

8. 引用数组的下标变量时，其下标值不能超出 DIM 语句定义的上界或下界，否则将出现超界错误信息。例如：

```
DIM X(15)
FOR I=1 TO 15
  FOR J=1 TO I+1
    IF X(J)<X(I) THEN
      ...
NEXT J,I
```

当外层循环 I 变化到 15 时，而内层循环将执行：X(16)<X(15)的比较。

系统会提示：Subscript out of range

7.5.4 一维数组的应用

● 一维数组的赋值

用 DIM 语句定义之后，系统自动将数值数组初始化为零，字符数组初始化为空 。利用数组处理数据时，必须在处理之前，把数据存放到数组中去，也就是说，数组元素在被引用之前，应当赋以必要的值。

对于一维数组的赋值，可在循环语句中，用 LET、INPUT、READ/DATA 三种方式输入数据。

1. 用 LET 语句赋值

一般用于数据比较特殊的数组元素，例如，数组元素全为 1，或数组元素是有规则的数列。例如：

①
```
OPTION BASE 1
DIM A(10)
FOR I=1 TO 10
    A(I)=1
NEXT I
```
该数组 A，十个元素都为 1

②
```
OPTION BASE 1
DIM A(10)
FOR I=1 TO 10
    A(I)=I
NEXT I
```
该数组 A，十个元素为 1—10

2. 用 INPUT 语句赋值

数据量不多，而又需要经常变动，使用较灵活时，可用这种方法，例如：

```
OPTION BASE 1
DIM A(10)
FOR I=1 TO 10
  INPUT A(I)
NEXT I
```

3. 用 READ/DATA 语句赋值

数据量大时，适合采用这种方法

```
OPTION BASE 1
DIM A(10)
FOR I=1 TO 10
  READ A(I)
NEXT I
DATA 8,5,13,21,4,17,23,2,15,9
```

● 一维数组的输出

数组需要输出到屏幕上显示时，要在循环语句中用 PRINT 语句。输出的格式由 PRINT 语句指定。例如：

```
1. FOR I=1 TO 10
     PRINT A(I)        '分行格式输出，每一个数组元素的值占一行，共占 10 行
   NEXT I
2. FOR I=1 TO 10
     PRINT A(I);       '紧凑格式输出，十个数组元素的值共占一行
   NEXT I
3. FOR I=1 TO 10
     PRINT A(I),       '分区格式输出，十个数组元素的值共占两行
   NEXT I
```

● 数组元素的引用

数组元素就是下标变量，它和简单变量的引用完全相同，可以被赋值，也可以出现在各种表达式中，例如：

```
A(1)=5                          'A(1)被赋以常数
A(I)<=A(J)                      '数组元素进行比较
A(I)=A(I+2)                     '数组元素相互赋值
T=A(J):A(J)=A(I):A(I)=T         '数组元素与简单变量的相互赋值。
```

注意:不能在程序中引用数组名。例如:A 是数组名,不能写成 A=5。

例:某比赛由 20 位评委评分,每位参赛者成绩的计算办法是:各评委评分中去掉一个最高分,去掉一个最低分,再计算余下的平均分

【算法】

1. 用一维数组 N 来存放 20 个评委的分数;

2. 对数组元素进行比较,找出最高分和最低分,同时对数组元素求和;找最大值和最小值的方法是用基准元素比较法,首先选定数组中某一元素为基准,作为最大者 MAX、最小者 MIN 的基准(本例中选数组第一个元素 N(1)作基准),然后把数组中每一元素和它们进行比较,谁比 MAX 大谁就代替 MAX,谁比 MIN 小谁就代替 MIN。最后必然是最大的数留在 MAX 中,最小的数留在 MIN 中。

3. 输出参赛者的成绩。

【源程序】

```
'Example QB7-19.bas
OPTION BASE 1
DIM N(20)
'Get the Values
FOR I = 1 TO 20
  INPUT N(I)
NEXT I
'Find the Maximun and Mininum and Suming
MAX = N(1): MIN = N(1): SUM = 0
FOR I = 1 TO 20
  SUM = SUM + N(I)
  IF N(I) > MAX THEN MAX = N(I)
  IF N(I) < MIN THEN MIN = N(I)
NEXT I
MEAN = (SUM - MAX - MIN) / (20 - 2)
'Output the result
PRINT "The Mean is:"; INT(MEAN * 100 + .5) / 100
END
```

例:把 10 个 100 以内的随机整数按由小到大的顺序排列出来

【算法】

1. 用随机函数产生 10 个不大于 100 的整数,并赋值给一个一维数组;

2. 把数组排序;

3. 输出已排序的数组。

这个算法的 1、2 步骤,可再进一步细化。

第一步细化为：

(1) 为了使每一次产生的随机数不相同，必须用 RANDOMIZE 打开随机数发生器。

(2) 定义一个数组存放随机整数，设为一维整数型，取名为 RAN，上界为 10，存放十个整数的存贮空间已够用了。

(3) 标准函数 RND(x)，产生(0,1)之间的随机数，现在要把它转换成不大于 100 的整数，所以要将此随机数扩大 100 倍，然后用取整函数取为整数，即用

INT(RND * 100)

第二步用选择排序法，其步骤如下：

(1) 将数组元素的第 1 个(RAN(1))依次与后面的九个元素比较，有谁比 RAN(1)还小就进行一次交换，把小的放在 RAN(1)中，这样把十个中的最小者就交换到第一个元素的位置上。

(2) 再将第二个元素 RAN(2)依次与后面的八个元素比较，和步骤(1)相同，把后 9 个中的最小者交换到第二个元素的位置上。

(3) 依次将第三、第四、…第九个元素重复上述步骤，最后就排序好了。这可以用一个循环来完成。

【源程序】

```
'Example QB7－20. bas
DEFINT A－Z
RANDOMIZE
DIM RAN(10)
'Product the Random Number
FOR I = 1 TO 10
  RAN(I) = INT(RND * 100)
  PRINT RAN(I);
NEXT I
PRINT
'Sorting in Compare Sort
FOR I = 1 TO 9
  FOR J = I + 1 TO 10
    IF RAN(J) < RAN(I) THEN SWAP RAN(J), RAN(I)
NEXT J
NEXT I
'Display the Sorted Array
FOR I = 1 TO 10
  PRINT RAN(I);
NEXT I
END
```

7.5.5 二维数组的应用

● 二维数组的赋值

二维数组的赋值和一维数组的赋值相似，采用双重循环的方式，用 LET、INPUT、READ/DATA 语句给每个数组元素赋值。

例：用赋值语句，给一个单位矩阵赋初值，单位矩阵是对角线上的元素为 1，其余元素为 0 的矩阵

```
OPTION BASE 1
DIM A(5,5)
FOR I=1 TO 5
  FOR J=1 TO 5
    IF I=J THEN
      A(I,J)=1
    ELSE
      A(I,J)=0
    ENDIF
NEXT J,I
```

例：用 INPUT 语句赋值：

```
DIM B(1 TO 5,1 TO 5)
FOR M=1 TO 5
    FOR N=1 TO 5
      INPUT B(M,N)
NEXT N,M
```

例：用 READ/DATA 语句赋值

```
DIM C(1 TO 10, 1 TO 2)
FOR J=LBOUND(C,1) TO UBOUND(C,1)
    FOR K=LBOUND(C,2) TO UBOUND(C,2)
      READ A(J,K)
NEXT K,J
DATA 2,4,6,8,9,12,14,16,18,19
DATA 1,3,5,7,9,11,13,15,17,21
```

● 二维数组的输出

二维数组的输出采用双重循环方式，用 PRINT 语句按一定格式输出每个数组元素，一般按行的顺序输出为好。

例：按行输出二维数组 X 的元素

$$X=\begin{vmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{vmatrix}$$

【源程序】

```
'outer loop control row
FOR I=1 TO 2
  FOR J=1 TO 3
    PRINT X(I,J),
  NEXT J
  PRINT
NEXT I
```

【运行结果】

```
 1  2  3
 4  5  6
```

7.6 过程：子程序和函数

7.6.1 FUNCTION 子程序

【格式】

FUNCTION Name[(a_1,a_2,...a_n)] [STATIC]
...
Name=e
[EXIT FUNCTION]
...
END FUNCTION

【说明】

1. 必须以关键字“FUNCTION”开头，以“END FUNCTION”结束。
2. Name　函数名，它的命名规则与变量名的命名规则相同，可以使用类型定义符。

它不仅是一个函数子程序的识别符，而且也存放函数的返回值，返回值的类型与函数名的类型一致。

3. Name=e 在函数子程序中要给该函数名赋值，才能把表达式 e 的值从子程序返回。函数名可以多次赋值，只有最后赋的值才被返回。函数名不能象普通变量一样参加运算或在算术表达式中引用。

4. $(a_1,a_2,...a_n)$ 是形式参数，简称形参，它可以是简单变量名、数组名，而不能是下标变量名。函数名后的形参是可选的，有多个形参时，各形参之间用逗号分隔。数组名后，必须跟一对括号。例如数组 A()。

5. EXIT FUNCTION 可选项。用它可从函数子程序中退出。

6. STATIC 可选项。用以指定调用子程序后，程序中所有变量的状态。

例：计算以十为底的对数和以 e 为底的自然对数

有如下换底公式：

lgX=ln(X)/ln(10)

在 QB 的标准函数中只有计算自然对数的函数 LOG(X)，没有以十为底的对数，现在可以编写一个函数子程序：

```
FUNCTION LOG10(X)
    LOG10=LOG(X)/LOG(10)
END FUNCTION
```

● FUNCTION 子程序的调用

1. 在调用程序的开头，使用 DECLARE 语句对过程名及其形参的个数、类型进行说明。

2. 象引用标准函数的方法那样调用函数子程序，它只能出现在表达式中，用实际参数（简称实参）代替函数子程序中的形参。

例：计算下式 Y 的值

$Y=(2\pi R+e^{-5})\lg 2x$

【源程序】

```
'Example QB7－23.bas
DECLARE FUNCTION LOG10 (X)
CONST PI = 3.14159
INPUT R, X
Y = (2 * PI * R + EXP(－5)) * LOG10(2 * X)
PRINT "Y="; Y
END
```

```
FUNCTION LOG10 (X)
LOG10 = LOG(X) / LOG(10)
END FUNCTION
```

【说明】

1. 在调用过程的程序开头，必须用 DECLARE 语句说明。如没有说明，在 QB 中，首次存盘时，系统会自动添加上该说明语句。

2. 实参和形参的个数相等、位置一一对应、类型相同，而形参和实参的名字可以不相同。

3. 形参是数组时，数组名后必须跟一对圆括号，调用时的实参数组名后也应跟一对圆括号。这样，形参数组的维数及下标的上、下界，将由对应的实参数组而定。

4. 调用函数子程序只能返回一个函数值，不能返回多个值。

5. 函数名后可以没有形参，称之为无参 FUNCTION 子程序。但在 DECLARE 说明中，函数名后仍然要用一对空括号表示。

7.6.2　SUB 子程序

SUB 子程序的定义和调用方法已在前面作了介绍，此处从略。

7.6.3　过程中的形实结合

● 形实结合的概念

在过程中定义的 SUB 子程序（或 FUNCTION 子程序）带有形参，实质上，它是实际参数的位置标志符，当它被调用时，实际参数就取代形参，执行子程序。这就叫形实结合，在形实结合时，必须遵循位置一一对应，形实的个数相等，类型相同。如图 7—1 中，第一个整型实参 A% 与第一个整型形参 P1% 结合，第二个单精度实参 B 与第二个单精度形参 P2 结合，第三个字符串实参“TEST”与第三个字符串形参 P3$ 结合。

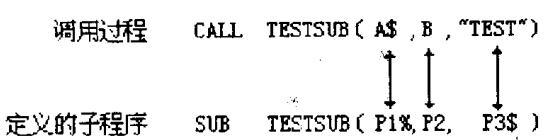


图 7—1　形实结合过程

中的形参可以是：

固定长字符串而外的各类型的简单变量名；

固定长字符串而外的各类型的数组名，名字后跟一对圆括号。

2. 实参表中的实参可以是：

· 常数

· 表达式
· 简单变量、下标变量
· 数组名，后跟一对圆括号

由于实参的种类多，形实结合时的具体情况有所不同，以下将分别进行讨论。

● 以常数和表达式作为形实结合的情况

例：

【源程序】

```
DECLARE PRTTRIG(N,X$)
    ...
CALL PRTTRIG (3,"*")
    ...
END

SUB PRTTRIG (N,X$)
FOR I=1 TO N
    PRINT TAB(31-I);
    FOR J=1 TO 2*I-1
      PRINT X$;
    NEXT J
    PRINT
NEXT I
END SUB
```

SUB PRTTRIG 子程序中，形实结合后，N 的值为 3，X$ 的值为"*"，执行程序后，打印出如下三行的三角形：

```
        *
      * * *
    * * * * *
```

这种形实结合的特点是：

1. 数字常数传送给数值形参，字符串传送给字符形参，传送的方向是实参传送至形参，执行过程中，不可能改变实参的值。这种传送方法，实质上是从存贮单元中取出存放的值，传送到分配给形参的存贮单元中去。如图 7—2 所示，这种方式称为按值传送。

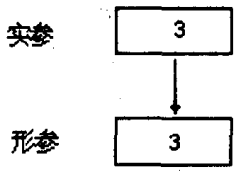


图 7—2 按值传送的过程

2. 数字常量的实参与相应的数值型形参不是同一类型时，常数被强制为与形参一样的类型。

3. 由常量和变量运算组合成的表达式传送过程与常数相同，首先是计算表达式的值，按值传送给形参，进行形实结合。

● 以简单变量作为形实结合的情况

例：计算三角形的面积

【源程序】

```
'Example QB7—24.bas
'Calculating Area of Triangle
DECLARE SUB Triga (a, b, c, s)
X = 4: Y = 7: Z = 5
Triga(X, Y, Z, a)
PRINT "Area="; a
END

SUB Triga (a,b,c,s)
IF a+b>c AND b+c>a AND c+a>b THEN
   p = a+b+c
   s = SQR(p * (p—a) * (p—b) * (p—c))
ELSE
   PRINT "Input data error"
   EXIT SUB
END IF
END SUB
```

变量的形实结合如图 7—3 所示：

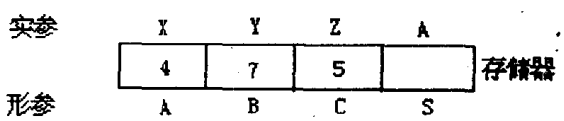


图 7—3 按地址传送的过程

当形实结合时，只是把实参存放的地址传送给形参，也就是形参和实参共同存取同一存贮单元，称为按地址传送方式，这样传送的特点是：

1. 传送是双向的，既可把实参的值传送给形参(称为输入)，也可以把形参中的值传送给实参(称为输出)。上一个子程序执行后，S 取得的值也带给了实参 A，故过程调用结束后，返回到调用程序去执行输出三角形面积时，A 的值就是 S 的值.

2. 在执行过程之后，实参的值可能被改变。

● 以数组名作为形实结合的情况

实参和形参都是数组时，数组名后必须跟一对空圆括号。

例：CALL test(NAME3(),I,J)

　　...

　　SUB test(ARRAY(),M,N)

数组的形实结合，也是按地址传送的。

例：

【源程序】

```
'Example QB7—25. bas
DECLARE SUB asign (Min, Max, p() AS INTEGER)
DECLARE SUB PrtA (Min, Max, p() AS INTEGER)
DIM Values(1 TO 5) AS INTEGER
CALL asign(1, 5, Values())
PrtA (1,5, Values( ))
NEXT i
END

SUB asign (Min, Max, p() AS INTEGER) STATIC
FOR i = Min TO Max
    p(i) = i * i
NEXT i
END SUB

SUB PrtA (Min, Max, p() AS INTEGER) STATIC
FOR i = Min TO Nax
    PRINT p(i)
NEXT i
END SUB
```

这个程序第一次调用子程序 ASIGN，给数组赋值，第三次调用子程序 PrtA，输出已赋值数组。实参数组由 DIM 语句确定上下界，形参数组的上下界由调用程序的实参决定。

7.6.4 DECLARE 语句

【格式】

DECLARE {FUNCTION| SUB} name 〔(a1,a2,.....an)〕

其中:过程名 name 和形参表(a1,a2,.....an)与定义 FUNCTION 子程序和 SUB 子程序的规定完全相同。

【功能】

使用 DECLARE 语句去检查实际参数与被调用过程中形参的个数和类型,确认是否一致。

【用法】

1. 在用 QB 集成环境下编程时,当程序存盘时,系统会把所有过程定义放到模块的末尾,而在模块的开头为每一个过程自动插入一条 DECLARE 语句。即使你的源程序开头没有使用 DECLARE 语句,存盘后,再调出来检查,就会发现系统已经给你加上去了。

但是,你的程序中包括有调用另外一个模块中定义的 FUNCTION 过程,QB 不在本模块产生 DECLARE 语句。因此,必须自己设置 DECLARE 语句。

此外,对于 Quick 库中的过程,也不自动产生 DECLARE 语句。

2. 在 QB 集成环境之外,使用别的文本编辑写程序,用 BC 和 LINK 命令编译,必须由用户把 DECLARE 语句加到调用程序的模块开头。

3. 当说明无参过程时,DECLARE 语句的过程名后要跟一对圆括号。

7.6.5 全局量与局部量

● 全局量与局部量的概念

在结构化程序设计中,程序可由多个模块构成,每一个模块内的代码,可分为模块级代码和过程级代码,因此在这些程序中出现的变量和常量在哪些模块级或代码级是有效的,即它的作用范围,这就产生了全局量与局部量的问题。

1. 全局量是在整个模块内都有效的变量或常量。

2. 局部量是只在一个模块内的过程中或模块级代码中有效的变量或常量。

3. 共享变量是只在一个模块内的各过程之间共享,或多模块程序的各模块之间共享。

● 全局变量与全局常量

全局变量是在模块代码中用带有 SHARED 选项的 DIM、REDIM 或 COMMON 语句来说明的变量。

全局常量是在模块中用 CONST 语句说明的常量。

例：

【源程序】

```
'Example QB7－26. bas
DECLARE SUB TestG ()
DIM SHARED X AS INTEGER
CONST T% = 5
X = 2
PRINT "Before call T * X="; T% * X
CALL TestG
PRINT "After call T * X="; T% * X
END

SUB TestG
    PRINT "In SUB T * X="; T% * X
    X = 3
END SUB
```

【运行结果】

```
Before call T * X= 10
In SUB T * X= 10
After call T * X= 15
```

由此可见，常量 T 在模块级代码和过程级代码都有效，在模块级变量 X＝2 时，进入子 程序后仍为 2，且当在子程序中赋值为 3 时，返回模块级时，X 也更新为 3。

● 局部变量与局部常量

凡未用带 SHARED 选项的 DIM、REDIM 或 COMMON 语句说明的变量都是局部变量，未用 CONST 语句说明的常量也是局部常量。在定义过程中使用 CONST 语句说明的常量也是局部常量。

例：

【源程序】

```
'Example QB7－27. bas
DECLARE SUB TestGn ()
DIM X AS INTEGER
    T% = 5
```

```
    X = 2
PRINT "Before call T * X="; T% * X; ",T="; T%

CALL TestGn
PRINT "After call T * X="; T% * X; ",T="; T%

END

SUB TestGn
    CONST T% = 10
    PRINT "In SUB T * X="; T% * X; ",T="; T%
    X = 3
END SUB
```

【运行结果】

```
Before call T * X= 10,T= 5
In SUB T * X= 0,T= 10
After call T * X= 10,T= 5
```

由上可见，在模块级代码中的变量 T 和 X 只在模块级起作用，在过程中无效，在模块级中 X 被赋值为 2，进入过程中，虽然有同名的变量 X，但并不取得其值，所以 T * X=0。这是两个不相同的变量，即有不同的存贮单元。同样在过程中的 T=5，X=3。从过程返回模块级后，它也不影响模块级原来 T 和 X 的值。正因为局部变量只有在限定的范围有效，即使是同名的变量也不会发生影响，这就对每一个模块，特别是各种子程序模块的编程带来了方便。只要在模块内定义的局部变量，不会受到别的模块的影响，由多人各自编写的程序，就不会受到相互的制约了。

7.7 字符串

7.7.1 有关字符串的基本概念

● 字符与字符串

1. 字符

指单个字母、数字、汉字或其它特殊符号。如 A、5、%等。

2. 字符串

用双引号("")括起来的一系列字符，称为字符串。如"ABCD"、"1234"、"中国"等。

3. 字符串长度

字符串中字符的个数，称为字符串长度。

4. 空串(null string)

指双引号内一个字符也没有的串，用""表示，其长度为 0。

5. 空格串(space string)

指双引号内包括的字符为空格字符(由空格键键入，ASCII 码为 32 的串)，称为空格串，用" "表示，空格要计算长度。

● 字符串常数与变量

1. 字符串常数又称常量串，有以下两种表示方法：

直接表示——用双引号括起来的字符串，如 "I am a Chinese!"

间接表示——用 CONST 语句来说明一个串的名字，在程序中使用该常量名，而不直接使用字符串。例如：

```
CONST MESSAGE="open the door"
PRINT MESSAGE
```

2. 字符串变量

字符串变量又称串变量。用以存贮字符串的值，其值在程序执行过程中可以被改变。定义字符串变量类型有三种方法：在变量名后加 $ 符号；使用 DEFSTR 语句来定义以某个 字母开头的变量为串变量；用 DIM 语句说明串变量类型。

3. 字符串的测长函数 LEN

【格式】

LEN (〈字符串表达式〉)。

【功能】

返回字符串的长度，即字符串表达式中字符的个数。

如"ABCD"的长度为 4。空格串" "长度为 1，""空串的长度为 0，"中国"的长度为 4，因为一个汉字占用两个字节。

【说明】

其中〈字符串表达式〉既可为字符串变量，又可为字符串，还可以是字符串变量和字符串连接符组成的表达式。

● 固定长度字符串变量与可变长度字符串变量

用 DIM 语句定义类型时，同时指定其长度，一经定义，其长度固定。说明方法如下：

```
DIM RST AS STRING * 10
```

字符串变量 RST 的长度最多就只能存 10 个字符，这称为固定长度字符串变量，用其它方法定义的字符串变量，则是可变长度字符串变量，可以存放的字符数最多为 32767。

7.7.2　字符串变量的赋值(LINE INPUT 语句)

【格式】

LINE INPUT[;][“〈提示字符〉”;]〈串变量〉

【功能】

从键盘上读入一整行字符(从提示字符之后到回车前的所有字符)。

【说明】

1. 执行此语句时,提示字符后不显示“?”。这与 INPUT 语句不同。

2. 对输入的字符没有限制,包括引号、逗号、空格等。

3. 如果 LINE INPUT 后紧跟一个分号,如:

```
LINE INPUT; "Enter your name?";A
```

那么按回车键结束输入后,光标不换行,仍保留在输入字符的同一行上。

例:

【源程序】

```
'Example QB7-28.bas
LINE INPUT "A$=?";A$
PRINT "* * * * * * * * * * *"
PRINT A$
LINE INPUT;"B$=";B$
PRINT "* * * * * * * * * * *"
PRINT B$
END
```

7.7.3　从键盘读入字符(INKEY$函数)

【格式】

INKEY$

【功能】

从键盘上读入一个字符

【说明】

1. 利用 INKEY$函数,程序可以随时对键盘进行监视搜索,以确定用户有没有打入什么字符或命令,如果没有,INKEY$返回空串,程序继续执行,如果有,则执行相应的操作。

2. 在程序设计中,INKEY$函数常被用来暂停程序的执行,等待用户按下任一键(或指定的键)后执行下一步操作。

例：

【源程序】

```
'Example QB7－29. bas
PRINT "PRESS SPACEBAR TO CONTINUE"
DO
LOOP WHILE INKEY$ =""
PRINT "PRESS ESC TO QUIT"
DO
    PRINT "123456789"
LOOP UNTIL INKEY$ =CHR$ (27)
END
```

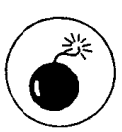
7.7.4　字符串的连接

字符串的连接就是把两个或多个字符串连接成一个新的字符串。

例：A1$ ＋A2$ ＋A3$，其中“＋”为连接运算符

【源程序】

```
'Xample QB7－30. bas
A$ ="Southwestern"
B$ ="Agricultural"
C$ ="University"
D$ =A$ ＋" "＋B$ ＋" "＋C$
PRINT D$
END
```



连接后的字符串长度不能大于 32767。

7.7.5　字符串的比较

在电脑内部，字符一般都是以 ASCII 码表示的。比较两个字符的大小，就是比较它们的 ASCII 码的大小，如"A"小于"B"。比较两个字符串的大小，其比较办法是将两个字符串从左至右逐个字符进行比较，直到出现不同字符为止，其字符串的大小就由这两个不同字符的 ASCII 码值而定。ASCII 码值大的字符所在的字符串比 ASCII 码值小的字符所在的字符串大。在 ASCII 码表中，常用字符的 ASCII 码的大小顺序为：

空字符(Blank)＜空格符(Space)＜数字(0－9)＜大写字母(A－Z)＜小写字母(a－

z)。两个字符串的比较,可以用关系运算符来完成,其结果为真或假。例如:

"doorman"<"doorwoman"　结果为真,因为前 4 个字符都相同,第 5 个字符"m"的 ASCII 码值小于"w"的 ASCII 码值。

"ASCII"<"ascii"　结果为真,因为大写字母小于小写字母。

"abc"="abc"　结果为真,因为长度相等,字符相同。

"aaa">"aa"　结果为真,因为第 3 个字符"a"大于空字符(blank)。

" ">""　结果为真,因为空格符大于空字符。

"32">"23"　结果为真,因为"3"的 ASCII 码大于"2"的 ASCII 码。

"32">"AB"　结果为假,因为数字的 ASCII 码小于字母 的 ASCII 码。

7.7.6　字符串的检索

【格式】

INSTR ([〈起始位置〉,],〈字符串 1〉,〈字符串 2〉)

【功能】

从起始位置开始,在字符串 1 中查找字符串 2 在字符串 1 中出现的位置。

若没有指定起始位置,则隐含为从字符串 1 的第一个字符起开始查找。

【说明】

1. 起始位置为 0—32767

2. 起始位置大于字符串的长度或者找不到字符串 2,则返回 0。如:

INSTR(5,"ABC","AB")的值为 0

INSTR(2,"ABC","AB")的值为 0

3. 从起始位置开始,字符串 2 在字符串 1 中不止出现一次时,则只返回第一次出现的位置。如:

INSTR(2,"ABABABC","AB")的值为 3。

例:

【源程序】

在 "the Quick BASIC is greater than the MS—BASIC"中,找出 the 的位置和数量。

```
'Example QB7—31.bas
String1$="the Quick BASIC is greater than the MS—BASIC"
String2$="the"
Start=1
matchnum=0
PRINT string1$
DO
  match=INSTR(start, string1$,string2$)
  IF match>0 THEN
```

```
    PRINT TAB(match); string2$
    Start=match+1
    matchnum=matchnum+1
    END IF
  LOOP UNTIL match=0
  PRINT "number of matches=";matchnum
  END
```

7.7.7 取子字符串

从一个字符串分割抽出的一组连续的字符所组成的新字符串，称为原来字符串的子字符串(substring)，简称子串，完成这一过程的操作，称为字符串的分割操作。如 A$="ABCD"，则"CD"就是 A$的一个子串，"A"，"AB"，"BC"等也是 A$的子串，即一个字符串有多个子串。

● 从左边开始取子串的函数

【格式】

LEFT$(X$,N)

【功能】

从 X$的左边第一个字符起，取出 N 个字符组成一个新字符串。

【说明】

1. 0≤N≤32767

2. 若 N≥X$的长度时，则得到原字符串；若 N=0，则取得一个空串 (null string)，长度为 0。

例：

【源程序】

```
'Example QB7-32.bas
A$="I love China"
B$=LEFT$(A$,1)
C$=LEFT$(A$,13)
D$=LEFT$(A$,0)
PRINT "B$=";B$;TAB(20);"Length=";LEN(B$)
PRINT "C$=";C$;TAB(20);"Length=";LEN(C$)
PRINT "D$=";D$;TAB(20);"Length=";LEN(D$)
END
```

● 从右边开始取子串的函数

【格式】

RIGHT$(X$,N)

【功能】

从字符串 X$ 的右边第一个字符开始向左数，取 N 个字符组成一个新串。

例：

【源程序】

```
'Example QB7-33.bas
A$="I love China"
B$=RIGHT$(A$,2)
C$=RIGHT$(A$,13)
D$=RIGHT$(A$,0)
PRINT "B$=";B$;TAB(20),"length=";LEN(B$)
PRINT "C$=";C$;TAB(20),"length=";LEN(C$)
PRINT "D$=";D$;TAB(20),"length=";LEN(D$)
END
```

● 从字符串的任意位置取子串的函数

【格式】

MID$(X$,M[,N])

【功能】

从 X$ 中第 M 个字符开始，取 N 个字符串组成一个子字符串。

【说明】

1. 1≤M≤32767，0≤N≤32767
2. 当 N=0 或 M 大于字符串的长度时，得空字符串(Null string)。
3. 当 N 省略或大于 X$ 的长度时，则返回从 M 开始的所有字符。

例：

【源程序】

```
'Example QB7-34.bas
A$="The people's Republic of China"
PRINT MID$(A$,5,6)
PRINT MID$(A$,26,5)
PRINT MID$(A$,14)
```

```
PRINT MID$(A$,26,30)
END
```

7.7.8 字符串的生成与字符串中字符的改变

● 生成字符串的函数 STRING$

【格式】

STRING$(number,string)

【功能】

返回一个由 number 个字符(该字符为 string 串中的第一个字符或 ASCII 码对应的字符)组成的字符串。

例:A$=STRING$(10,"A")　　则 A$为"AAAAAAAAAA"
　　A$=STRING$(3,65)　　则 A$为"AAA"
　　A$=STRING$(3,"BCD")　　则 A$为"BBB"

● SPACE$函数

【格式】

SPACE$(N)

【功能】

产生 N 个空格组成的空格字符串。0≤N≤32767。此函数在数据对齐输出时非常有用。

例:

【源程序】

```
'Example QB7-35.bas
FOR I=1 TO 3
READ X$
A$=SPACE$(11-LEN(X$))+X$ '尾对齐
PRINT I,A$
NEXT I
DATA Mathmatics, Computer, Chemistry
END
```

7.7.9 字符串与数的相互转换

QB 不允许将数值赋给一个串变量,也不允许将字符串赋给数值变量,否则将会产生

数据类型不匹配的错误。QB 提供了字符与 ASCII 码的相互转换以及数字和字符串的相互转换的函数 ASC 函数及 CHR 函数。

● ASC 函数

【格式】

ASC(〈字符串〉)

【功能】

将字符串中的第一个字符转换成对应的 ASCII 码，并以十进制数表示。

【说明】

如果字符串为空串，则出现“非法函数调用”的语法错误。

例：ASC("A")　　函数的值为 65

　　ASC("BC")　　函数的值为 66

● CHR $

【格式】

CHR $(〈ASCII 码〉)

【功能】

将该 ASCII 码转换为所对应的字符。

【说明】

CHR $ 通常用于向屏幕或打印机发送一个特殊字符，例如：

PRINT CHR $(02)执行该句后，清屏并把光标移至左上角复位位置。

QB 规定，字符串中不能带有双引号(")，但 CHR $ 可以使字符串中包含双引号。

● STR $ 函数

【格式】

STR $(〈数字表达式〉)

【功能】

返回一个由数字表达式构成的字符串。

【说明】

如果数字表达式结果是正数，那么 STR $ 函数返回的字符串中会有一个空格。该空格是正号省略留下的。

● VAL 函数

【格式】

VAL(〈字符串表达式〉)

【功能】

返回一个由字符串表达式转换而成的数值。

【说明】

1. 如果字符串中的第一个字符不是数字,那么该函数的返回值为 0。

2. VAL 函数的字符串自动跳过字符串开头的空格。

3. VAL 函数的字符串中部含有空格或其它非数字字符,则自动忽略此字符之后的所有字符。

例:VAL("−5")　　函数的值为−5

VAL("3.14159")　　函数的值为 3.14159

VAL("1.955E3")　　函数的值为 1955

VAL(" −33.3")　　函数的值为−33.3

VAL("20 * 40")　　函数的值为 20

7.8 文件

7.8.1 文件的基本概念

● 什么是文件

文件是存放在外部存储介质上的有关联数据的集合。把编写的 Quick BASIC 源程序,取一个名字,存放在磁盘上,就是一个程序文件;如果把一批数据,例如一个学生的学科成绩,也取一个名字存放到磁盘上,就是一个数据文件。

文件是由记录(Record)组成的,一个记录是由一个或多个数据项(Data item)组成,数据项又叫字段(有的又叫域)。它是最基本的不可分的数据单位。例如,下面是一张学生成绩表格,可以构成一个数据文件。

学生成绩表

学　号	姓　名	政治	语文	数学
940101	李　鸣	83	92	88
940102	张　永	72	82	73
…	…	…	…	…
940130	周　青	90	89	94

假定有 30 个学生，这个文件就由 30 个相同结构的记录组成，每一个学生的学号、姓名、政治、语文、数学五个项，每一项就是一个字段。

程序文件的记录一般只有一个字段，就是一个字符组。比如 Quick BASIC 源程序中的一个程序行就是一个记录。

因此，我们也可以说文件是由大量性质相同的记录组成的集合。

● 文件的命名

文件名是用户为文件起的名字，是文件的标志。使用时，通过文件名来实现对文件的存取。

1. 文件名的格式

〈文件名〉[〈.扩展名〉]

文件名：由 1～8 个字符组成。文件名可以是如下字符：

英文字母：a～z，A～Z

数字：0～9

特殊字符：(，)，{，}，@，#，$，%，!，—，_，/，^，&，～

扩展名：由 0～3 个字符组成，排在文件名的后面，用小数点分开，用于区别文件类型。

如：A1.BAS，FANG.BAS，1.BAT 等都是合法的文件名。

2. 说明

(1)文件名前可以用驱动器号和路径名，以便访问不是当前驱动器和路径下的文件。

例如：　B:\RS\STUD.DAT

(2)给文件命名时最好选用与文件内容或性质相关的字符，以便见名知义，知道该文件属于哪一类，大致是什么内容。

(3)文件名与扩展名的总字符数不能大于 11，如大于 11，取前 8 个字符作为文件名，插入一个小数点，再取后面的 3 个字符作为扩展名。

(4)文件名中的大小写无区别。

● 文件的分类

1. 根据文件中存放内容的功能分类

· 程序文件(Program file)

由计算机中的可执行程序或高级语言编写的源程序组成的文件。

· 数据文件(Data file)

存放数据的文件。如：学生成绩、职工工资等。

2. 根据数据访问方式及结构分类

· 顺序文件(Sequential File)

这是最简单的文件结构，文件中的数据是按照从小到大(从 0 号记录开始)顺序存贮

的。读取某个数据时，必须从头逐个记录地进行查找，直到查到为止。顺序文件以 ASCII 码方式存贮。这种文件占用空间大，修改操作较复杂。

· 随机文件(Random Access File)

随机文件中的每个记录均有固定的长度，文件中的数据是按记录号写到磁盘上去的，读取数据时，也是按记录号读取。随机文件以二进制格式存贮。这种文件访问数据速度快，更新数据容易，能同时进行输入和输出操作，比顺序文件占用空间少。

3. 根据数据的存贮形式分类

· ASCII 文件

数据以 ASCII 码方式存放在磁盘中。

· 二进制文件

数据以二进制方式存放在磁盘中。

4. 根据数据的存贮介质分类

· 磁盘文件

数据存放在磁盘上。

· I/O 设备文件

QB 将进行输入输出的外部设备也象磁盘上的数据文件一样看待，统称为设备文件。如：键盘、显示器、打印机等。

● 文件的输入/输出

文件的输入/输出都是相对于计算机的内存缓冲区而言的。所谓文件输入，就是把文件从磁盘调入内存缓冲区，这与我们平时所说的读文件或读磁盘是一个意思。所谓文件输出，就是把内存缓冲区中的内容存放到磁盘文件中去，也可以称为写文件或写磁盘。

● 文件长度与文件的指针

1. 文件长度

指一个文件中字符的个数，用字节数来表示。

2. 文件指针

文件指针是用以指示文件中当前要读写数据的位置。文件一经打开，文件指针指在文件的开始位置。文件的读写都是通过文件指针来完成的。

7.8.2 用户自定义数据类型

前面，我们讲到了一些基本数据类型，如整型(%)，单精度型(!)，双精度型(#)，长整型(&)及字符串型($)等简单类型，但在现实生活中，往往又有一些属性同属于一个属主，而这些属性的类型又各不相同，我们又希望将该属主的所有属性做为一个整体来考虑，这就是包含多属性的记录。如：学生记录就大致应该包括姓名、性别、年龄、学习成绩等

四个字段，这四个字段的数据类型不相一致。为了解决这类问题，QB 允许用户自定这样一种数据类型，即在同一变量名下，能存贮一组不同类型的数据值，这种数据结构在 QB 中称用户自定义类型。

● 用户自定义类型的定义规则

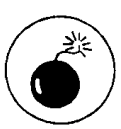
1. 用 TYPE—END TYPE 语句定义数据类型

【格式】

TYPE 〈数据类型名〉
〈成员名 1〉 AS 〈类型〉
〈成员名 2〉 AS 〈类型〉
…… ……
〈成员名 n〉 AS 〈类型〉
END TYPE

【说明】

(1)必须以 TYPE 开头，以 END TYPE 结尾。
(2)数据类型名　表示用户定义数据类型的名字，命名规则同变量名。
(3)成员名　表示该用户定义数据类型中的各成员名，命名规则同变量名。
(4)类型　表示各成员的数据类型，可以是 INTEGER(整型)，LONG(长整型)，DOUBLE(双精度)，SINGLE(单精度)，STRING * N(固定长字符串)或者其它数据类型。



TYPE—END TYPE 语句只允许固定长字符串，不允许可变长字符串。

例：将学生档案中的各字段建立一个名叫 PUP 的用户定义类型

```
TYPE PUP
  No AS LONG
  Nam AS STRING * 15
  Sex AS STRING * 1
  Age AS INTEGER
  Ymd AS STRING * 8
END TYPE
```

2. 用 DIM、REDIM、COMMON 等语句说明用户定义类型变量，以下我们简称记录变量。

【格式】

DIM〈记录变量名〉 AS 〈用户数据类型名〉

【说明】

记录变量属于其后指定的数据类型。

记录变量名也可以是数组，由此建立的数组称为记录数组。

例如：

DIM REC AS PUP 说明 REC 是一个属于 PUP 类型的记录变量。

DIM REC(100) AS PUP 定义包含有 100 个元素的数组，每一个数组元素都属于 PUP 类型。

● 记录变量的引用

记录变量定义后，其中每一个成员的表示方法为：

记录变量名.成员名

例如上例学生档案中，一个记录的 5 个字段表示如下：

Rec.no

Rec.nam

Rec.sex

Rec.age

Rec.ymd

例：在学生档案中，要求包括学号、性别、年龄、及出生年月日，请定义一个用户定义数据类型，然后输入两个学生的情况，并在屏幕上显示出来。

【源程序】

```
'Example QB7－36.bas
TYPE PUP
  No AS LONG
  Nam AS STRING * 15
  Sex AS STRING * 1
  Age AS INTEGER
  Ymd AS STRING * 8
END TYPE
DIM REC(10) AS PUP
FOR i = 1 TO 2
  READ REC(i).No, REC(i).nam, REC(i).sex, REC(i).age, REC(i).ymd
  PRINT TAB(1); REC(i).NO; TAB(10); REC(i).nam; TAB(25);
  PRINT REC(i).sex; TAB(30); REC(i).age; TAB(40); REC(i).ymd
NEXT i
DATA 930101,ZhangShan,m,19,74－02－28
DATA 930102,LiShi,w,18,75－02－27
END
```

7.8.3 顺序文件

访问顺序文件，通常有三个步骤：打开文件，读/写数据和关闭文件。

● 顺序文件的打开

打开文件就如我们记笔记一样，必须先准备好笔记本，然后才能记笔记。

打开文件用 OPEN 语句。

【格式】

OPEN〈文件名〉FOR〈模式 1〉[ACCESS〈模式 2〉]〈AS #文件号码〉

【参数】

1. 模式 1　为下列三种形式之一：

OUTPUT　建立一个新文件，以备将数据从内存中写入磁盘文件中。

APPEND　若文件已经存在，则打开该文件，将数据写到该文件的尾端，若没有该文件，则建立一个以文件名为名的新文件。

INPUT　打开一个已存在的文件，以备将磁盘文件中的数据读入内存中。

2. 模式 2　为下列三种形式之一：

READ　打开只读文件。

WRITE　打开只写文件。

READ WRITE 打开可读写文件，仅用于顺序文件的 APPEND 方式。

3. 文件号码：1～255 之间的一个整数。在内存中，文件是通过缓冲区与外部设备交换信息的，文件号码是系统给被打开的文件分配的缓冲区编号（又称缓冲号或通道），并把文件号码与文件名联系起来，在以后的读写操作中，就用文件号码来代替文件名。

【功能】

允许对文件进行输入或输出。

【说明】

格式中的选项 ACCESS，主要用于文件共享环境，本书暂不选用此项。

例：

1. 建立新的顺序文件，以便数据被写到文件中，或者打开原有的数据文件，使新数据取代文件中原有的数据。例如：

```
OPEN "BOOK.DAT" FOR OUTPUT AS #1
```

2. 打开原有的数据文件，使新数据增加到文件末尾，而不破坏文件中原有的数据。例如：

```
OPEN "BOOK.DAT" FOR APPEND AS #1
```

如果给定的文件名在当前目录中不存在，这种方式也可以用来建立一个新的顺序文件。

3. 打开已经存在的顺序文件，使其中的数据可以被读出。例如：

OPEN "BOOK.DAT" FOR INPUT AS #1

● 顺序文件的写入

QB 通过 WRITE #，PRINT #和 PRINT USING #这三个语句向用 OUTPUT 或 APPEND 方式打开的顺序文件写数据。

1. WRITE #语句

【格式】

WRITE #〈文件号〉,〈表达式表〉

【功能】

将表达式表中的多项数据写入到与文件号相联系的磁盘文件中。

【说明】

表达式表中各项既可以为串变量，也可以为数字变量或常数，各项之间用逗号隔开。

每执行一次 WRITE #语句，就写入一个记录，并附加一个回车符，表示一个记录已写完。

例如：WRITE #1,A$,B$,C,D,123

2. PRINT #语句

格式、功能、说明同 WRITE #语句。

3. PRINT USING #语句

【格式】

PRINT #〈文件号〉,USING〈格式表达式;〉〈表达式清单〉

【功能】

将表达式清单中的数据按格式表达式写入到磁盘文件中。

【说明】

格式表达式由格式字符组成，格式字符及意义与 PRINT USING 语句中的格式符意义完全相同。

例如：PRINT #1, USING "#.##";A,1.256

● 顺序文件的读出

1. INPUT #语句

【格式】

INPUT #〈文件号〉,〈变量表〉

【功能】

从顺序文件中读取数据，并把这些数据赋给变量。

【说明】

变量表就是一系列变量，变量可以是字符串变量或数字变量，也可以是数组元素，变

量的类型必须与文件中数据的类型相同，各变量之间用逗号隔开，对于数字变量，INPUT #语句忽略先导空格和回车换行符，而把遇到的第一个非空格和回车换行符作为数字的开始。在数字后遇到空格或回车换行符时，则认为数字结束。

例如：INPUT #1,A$,B,C

2. LINE INPUT #语句

【格式】

LINE INPUT #〈文件号〉,〈串变量〉

【功能】

从顺序文件中读取一个完整的行(行以回车换行符为结束符)，赋给一个字符串变量。

【说明】

INPUT #和 LINE INPUT #的功能相似，只不过 INPUT #中语句读出的是文件的数据项，而 LINE INPUT #语句读出的是文件的一个完整行。

例如：LINE INPUT #1,A$

3. EOF 函数

【格式】

EOF(〈文件号〉)

【功能】

如果文件指针到达了顺序文件的末端，则 EOF 函数返回数值－1(真)，否则返回数值为 0(假)。

● 文件的关闭

文件使用完毕后，必须要关闭，否则就会引起数据丢失。关闭文件用 CLOSE 语句。

【格式】

CLOSE〔#〈文件号〉〕

【功能】

将缓冲区中的数据写入文件中，结束对文件的输入/输出，关闭文件通道。

【说明】

如果指定文件号，则关闭文件号相联系的文件，否则关闭全部打开的文件。

CLOSE 语句执行后，文件和文件号的对应关系就结束了，因此，该文件号被释放，可以用该文件号来打开另一个文件 。

对一个以顺序输出方式打开的文件，使用 CLOSE 语句后，它将最后输出缓冲区的内容写入文件，同时在文件尾加上文件结束标志，有了它，EOF 函数才能发挥作用。

CLOSE 语句释放与被关闭文件有关的全部缓冲区空间。

除 CLOSE 语句外，END 语句可以自动关闭全部文件。

例如：

```
CLOSE #1,#2,#3        '关闭 1 号,2 号,3 号文件
CLOSE                 '关闭全部打开的文件
```

● 顺序文件的使用

1. 顺序文件的建立

例：有一张清单，上面写着三本书的书名和书价。现在要求建立一顺序文件，将该清单存贮到一个文件中

书　　名	销售价格(元)
The art of Computer Programing	3.75
Programmer's Problem Solver for IBM—PC	7.00
Software Debugging for Micro Computer	1.65

【源程序】

```
Example QB7—37.bas
OPEN "book.dat" FOR OUTPUT AS #1          '建 BOOK.DAT 顺序文件
DO
INPUT "name of book"; Name $              '输入书名
IF UCASE $ (Name $ ) = "END" THEN         '书名为 END 则结束输入
    EXIT DO
END IF
INPUT "price"; price                      '输入价格
WRITE #1, Name $, price                   '将记录写入文件
LOOP
CLOSE #1                                  '关闭文件
END
```

程序运行后，按照屏幕提示将以上三本书依次输入，键入 END 后程序终止运行，顺序文件 BOOK.DAT 已经建立在当前盘上了。为了核实该文件的内容可以用 DOS 命令 TYPE 显示。

2. 顺序文件的读取

例：用 INPUT # 语句编一程序将顺序文件"BOOK.DAT"的内容显示出来

```
'Example QB7—38.BAS
CLS
```

```
OPEN "book.dat" FOR INPUT AS #1        '打开文件，准备读出
DO WHILE NOT EOF(1)
  INPUT #1, Name$, price               '读入一条记录
  PRINT Name$, price                   '显示一条记录
LOOP
CLOSE #1
END
```

7.8.4 随机文件

随机文件的输入输出是按随机方式进行的，它可以直接对文件中任何位置的记录进行存取，因此，使用的灵活性和存取速度都比顺序文件有较大的改善。

随机文件中记录的存贮与顺序文件有很大的差别，每个随机存取记录被定义为固定的长度，记录中的每个字段也被定义为固定的长度，这样固定的长度确定了记录或每个字段的起始或结束，它没有用于分隔字段的逗号和用于分隔记录的回车换行符。随机文件的数据是以压缩二进制格式存贮在磁盘上的。

例如，数字 15000，在顺序文件中，每一位占一个字节，共占 5 个字节，而在随机文件中，15000 是一个整型数，仅占两个字节。

● 随机文件的打开

【格式】

OPEN〈文件名〉〔FOR RANDOM〕AS #〈文件号〉〔LEN＝〈记录长度〉〕

其中：

〈文件名〉和〈文件号〉和顺序文件打开语句相同

FOR RANDAM　用以指定随机文件，可以省略。

LEN＝〈记录长度〉　用以指定随机文件的记录长度，它是记录的各个字段所占的字节数总和。〈记录长度〉是一个整数表达式，最大不超过 32767。省略 LEN，记录长度缺省值为 128 字节。

【功能】

打开一个随机文件。

例如：OPEN "X.ABC" FOR RANDOM AS #1 LEN＝15

表示将磁盘文件"X.ABC"以随机方式打开，它的记录长度为 15 个字节。

● 随机文件的输入/输出

1. 随机文件的输入

使用 GET #语句完成随机文件的输入。

【格式】

GET #〈文件号〉[,〈记录号〉][,〈输入变量〉]

【功能】

将文件中记录号指示的记录内容读入输入变量中。

2. 随机文件的输出

使用 PUT # 语句完成随机文件的输出。

【格式】

PUT #〈文件号〉[,[〈记录号〉][,〈输出变量〉]]

【功能】

将输出变量的值按记录号写到与文件号相关的文件中。

● 随机文件的建立

建立随机文件，必须首先定义记录内容各字段的类型和字段长度。QB 提供了以下两种定义方法。

1. Field 语句

【格式】

Field [#]文件号,〈字段宽的字节数〉AS〈串变量〉…

例如:Field #1,20 AS A$,30 AS B$,15 AS C$

这样定义的记录只允许以字符串类型存取，因此，在写入磁盘文件之前，数值类型的数据必须转换成字符串型(用专门的转换函数 MKI$、MKS$、MKD$)；相反，从磁盘读出文件时又得将它还原为数值型数据(也用专门的转换函数 CVI、CVS、CVD)。

2. 用 TYPE…END TYPE 定义一个用户定义类型的记录。

这样定义的记录，存取文件时就不需要进行类型转换。因而简单方便得多，本书只介绍这一种方法。

建立一个随机文件的步骤是：

(1)用 TYPE—END TYPE 定义一个记录；

(2)用 OPEN 语句打开文件，并指定每个记录的长度；

(3)获取一个新记录，用 PUT # 语句，写入到该文件中；

(4)记录存放完以后，关闭文件。

例:按下表所示的档案数据建立一个随机文件

学号	姓 名	性别	年龄	入学时间
92001	Li Fang	M	18	92—08—30
92002	Liu Bin	M	19	92—08—29
92003	Yang Mei	W	18	92—08—29

首先，我们定义一个用户数据类型 PUP，然后再说明一个 PUP 类型的记录变量 rec。

【源程序】

```
'Example QB7—39.bas
```

```
TYPE PUP
    no AS LONG
    nam AS STRING * 15
    sex AS STRING * 1
    age AS INTEGER
    ymd AS STRING * 8
END TYPE
DIM rec AS PUP
CLS
OPEN "student.dat" FOR RANDOM AS #1 LEN = LEN(rec)
INPUT "number of students"; n
FOR i = 1 TO n
    INPUT "number"; rec.no
    INPUT "name"; rec.nam
    INPUT "sex"; rec.sex
    INPUT "age"; rec.age
    INPUT "date"; rec.ymd
    PUT #1, i, rec
NEXT i
CLOSE #1
END
```

● 随机文件的读取

例：将上例建立的文件 STUDENT.DAT 中的数据显示出来

```
'Example QB7—40.bas
TYPE PUP
    no AS LONG
    nam AS STRING * 15
    sex AS STRING * 1
    age AS INTEGER
    ymd AS STRING * 8
END TYPE
DIM rec AS PUP
CLS
OPEN "student.dat" FOR RANDOM AS #1 LEN = LEN(rec)
recordnum = LOF(1) / LEN(rec)
r = 0
DO WHILE r < recordnum
```

```
    r = r + 1
    GET #1, r, rec
    PRINT TAB(2); rec.no; TAB(12); rec.nam; TAB(30); rec.sex;
    PRINT TAB(38); rec.age; TAB(46); rec.ymd
LOOP
CLOSE #1
END
```

语句 recordnum=LOF(1)/LEN(rec)计算文件中现有的记录个数。其中 LOF 函数计算"STUDENT.DAT"文件的总字节数，如果"STUDENT.DAT"是个新文件或空文件，则 LOF(1)返回 0. LEN(REC)计算一个记录的字节数，因此，记录数＝文件总字节数/一个记录的字节数

显然，这个公式对顺序文件是不适合的，因为顺序文件记录长度不相同。请同学们思考怎样编一个程序来计算顺序文件的记录数。

7.9 图形

7.9.1 电脑作图的基本知识

● PC 屏幕显示模式

显示模式，就是指电脑输出的信息在显示屏幕上显示的方式。在 PC 中有两种显示模式，即：字符显示模式和图形显示模式。显示模式由显示器和适配器的组合决定 PC 主要配有以下几类显示适配器：

MDA ：Monochrome Display Adapter	单色显示适配器
CGA ：Color Graphics Adapter	彩色图形适配器
EGA ：Enhanced Graphics Adapter	增强图形适配器
VGA ：Video Graphics Array	视频图形阵列
MCGA：Multicolor Graphics Array	多色图形阵列

MDA 只能配单色显示器，只支持字符显示模式。其余四种配彩色显示器，能支持两种显示模式，由于显示器和适配器的特性差异，它们有多种组合，这在 SCREEN 语句中详讲。

本书以 CGA 适配器的彩色显示器作为硬件基础来介绍作图功能。

1. 字符显示模式

字符显示模式又称文本模式或 TEXT 模式，在该模式下，屏幕上显示的最小单位是字符，屏幕上的显示位置用字符坐标系来描述。这种坐标系以屏幕的左上角为坐标原点，其坐标为(1,1)。水平方向为 X 轴，垂直方向为 Y 轴，文本方式有 40 列文本方式和 80 列文本方式，即一行中显示 40 个字符或 80 个字符。如图 7—4 所示。

图 7—4　字符显示方式屏幕

一个字符用 2 个字节来表示，第一个字节为字符本身，第二个字节为显示属性，所以，40 列文本显示方式需 2K 字节显示缓冲区，80 列文本显示方式需 4K 字节显示缓冲区。在显示时，一般把屏幕缓冲区分为若干页来使用，分页写入显示内容。CGA 有 16K 字节的屏幕缓冲区，因此 40 列文本方式时分为 8 页，编号为 0—7；80 列文本方式时分为 4 页，编号为 0—3。

2. 图形显示模式

在图形模式下，屏幕上显示的最小单位是显象管的扫描点，通常称之为象素(PIXEL)，又称像元，它是图形的基本元素，一般称为点。每一屏幕由若干条水平扫描线组成，每一扫描线又由若干个扫描点组成。通常把屏幕上象素的个数叫做分辨率。例如 320×200(点×线)称为中分辨率，640×200 称为高分辨率。在图形方式下，每个象素在屏幕上的显示位置用点坐标系来描述，这种坐标系以屏幕的左上角为坐标原点，原点坐标为(0,0)，水平方向为 X 轴，垂直方向为 Y 轴，对于 CGA 来说，QB 以两种不同的方式绘图，即中分辨率和高分辨率。如图 7—5 所示。

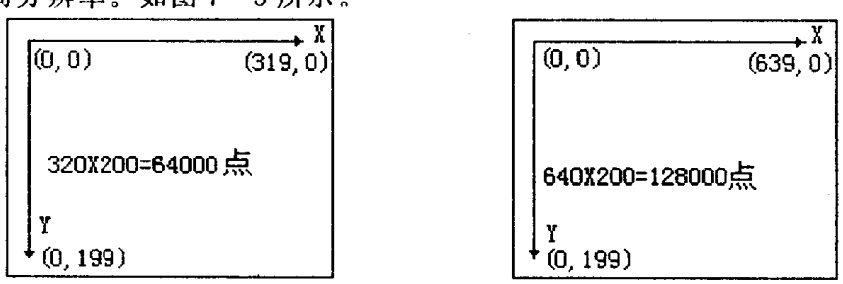


图 7—5　图形显示方式屏幕

在图形方式下仍可以显示文本字符。其中，高分辨率方式对应 80 列文本，中分辨率方式对应 40 列文本。

● 图形屏幕坐标

绘图语句中的坐标数据可以用两种形式给出，一种是绝对坐标，另一种是相对坐标。

1. 绝对坐标

参考点为点坐标系的原点(0,0),X、Y 只能取规定范围内的正整数。

2. 相对坐标

参考点为光标现行位置,即每一次绘图后,光标所停留的位置。在相对坐标中,X、Y 是相对于参考点的 X 方向和 Y 方向的位移量。既可为正整数,也可以为负整数。在 QB 中,相对坐标通过绘图语句中的选择项 STEP 来标识。

7.9.2 有关屏幕控制的语句

● CLS 语句

【格式】

CLS [〈0|1|2〉]

【功能】

清除屏幕上的全部或部分信息。

【说明】

CLS 0　清除屏幕上的全部文本和图形。

CLS 1　如果执行过 VIEW 语句,则仅清除图形视区,否则清除整个屏幕。

CLS 2　仅清除文本视区。

CLS　清除活动显示区(当前显示区),如果图形视区是活动的,则清除图形视区,否则清除文本视区。

CLS 语句在清除信息的同时,在图形模式下,将光点置于屏幕中心,在文本模式下,将光标返回到屏幕的左上角。

● 光标定位语句——LOCATE 语句

【格式】

LOCATE 〈行坐标〉,〈列坐标〉

【功能】

将光标移到指定的屏幕位置。

【说明】

语句中的行坐标或列坐标可以缺省,缺省时,以光标最后一次的行或列位置为当前的行或列的位置。

```
例如:LOCATE 1,1      '把光标移到屏幕左上角的起点位置;
      LOCATE ,5       '将光标移到当前行的第 5 列;
      LOCATE 5        '将光标移到当前列的第 5 行;
```

● SCREEN 语句

【格式】

SCREEN〈模式代码〉[,〈彩色开关〉][,〈当前页〉,〈可见页〉]

【功能】

设置屏幕显示模式

【说明】

1. 模式代码

设置显示方式代码，共有 10 种方式，取值为 0,1,2,7,8,9,10,11,12,13，每个代码值表示不同的显示方式。这主要根据所用的显示器和显示适配器而定，现将常用的 0,1,2 方式介绍于下。

0—— 文本方式，也是开机后的缺省方式。对 8×8 点阵字符可显示的文本大小为 40(字符或列)×25(行)，40×43，40×50，80×25，80×43 或 80 ×50(EGA 或 VGA 适配卡，字符可以是 8×14，9×14 或 9×16 的点阵)几种。2 种属性，每种有 16 种颜色，EGA 卡有 16 种属性，每种也是 16 种颜色。

1——中分辨率绘图方式，320×200 个象素点。8×8 点阵字符可显示为 40×25，EGA 卡或 VGA 卡有 4 种属性，每种有 16 种颜色，此方式支持 CGA，EGA，VGA 或 MCGA 卡。

2——高分辨率绘图方式，640×200 象素点。8×8 点阵字符可显示为 80×25，EGA 卡或 VGA 卡有 2 种属性，每种有 16 种颜色，此方式支持 CGA，EGA，VGA 或 MCGA 卡。

2. 彩色开关

是一个数值表达式，其值在 0—255 范围内。当该值为非 0 时，显示黑色图象，当该值为 0 时，显示彩色图象。当“模式代码”为 0 时，彩色开关的意义颠倒。当“模式代码”为 2 以上时，忽略“彩色开关”。

3. 当前页

是一个数值表达式，表示输出显示信息时被写入的页号。主要用于页式动画技术。

4. 可见页

是一个数值表达式，表示正在显示的屏幕页号。主要用于页式动画技术。

```
例:SCREEN 0          '置屏幕为文本文式
   SCREEN 1,0        '置屏幕为中分辨率图形方式,彩色显示
   SCREEN 2,1        '置屏幕为高分辨率图形方式,黑白显示
```

● 屏幕颜色设置语句——COLOR 语句

【格式】

COLOR 语句在不同的屏幕方式下使用不同的格式，常见的有：

1. 在中分辨率图形方式下颜色的选择：

COLOR〔前景色〕〔,背景色〕〔,边色〕

2. 在文本方式下颜色的选择：

COLOR〔背景色〕〔,调色板〕

【功能】

设置显示图形时屏幕的颜色

【说明】

前景色——指定当前属性下字符或图形的显示颜色。

背景色——指定当前属性下屏幕背景的颜色。

调色板——又称调色器，指定当前颜色的属性，取值范围是 0—255 的整数，其颜色属性有两种，奇数为一种属性，偶数为另一种属性。

为奇数选择 1 号调色板，为偶数选择 0 号调色板。两个调色板对于 SCREEN 1 方式，各有 4 种颜色，因此，作图语句中的彩色码取值为 0、1、2、3 四个与之对应，共有 8 种颜色可供选择，见表 7—5 和表 7—6。这样，用 COLOR 语句可对屏幕底色的 16 种颜色任选其一，还可用不同的调色板来改变前景的颜色。

COLOR 语句缺省某参数就表示保持原来的底色或调色板不变。使用 SCREEN 1 进入中分辨率彩色图形模式时，其初态是：底色为白色，调板为 1 号。

```
例如:SCREEN 1 : COLOR 0,4        '底色为黑色,0 号配色器
     LINE (5,5)—(25,25),3        '彩色码 3 对应的线条为黄色
     SCREEN 1 : COLOR 1,1        '底色为蓝色,1 号配色器
     LINE (5,5)—5,25),3          '彩色码 3 对应的线条为白色
```

表 7—5　分辨率图形式方式下的背景颜色

背景颜色号	颜　色	背景颜色号	颜　色
0	Black(黑)	8	Dark Grey(深灰)
1	Blue(蓝)	9	Light Blue(浅兰)
2	Green(绿)	10	Light Green(浅绿)
3	Cyan(青蓝)	11	Light Cyan(浅青)
4	Red(红)	12	Light Red(淡红)
5	Magenta(洋红)	13	Light Magenta(淡洋红)
6	Brown(棕)	14	Light Yellow(浅黄)
7	Light Grey(浅灰)	15	White(白)

表7—6　文本方式下的背影颜色

彩色码	配色器0	配色器1
0	当前背景色	当前背景色
1	绿	青
2	红	洋红
3	黄	白

7.9.3　画点语句

【格式】

PSET〔STEP〕(X,Y)〔,〈彩色码〉〕

PRESET〔STEP〕(X,Y)〔,〈彩色码〉〕

【功能】

PSET 按前景颜色在指定坐标画点

PRESET 按背景颜色在指定坐标画点

【说明】

1. 如果坐标的值超出了屏幕坐标系统所允许的范围，本语句不起作用。

2. 彩色码就是作图用的颜色代码，彩色码缺省时，PSET 用前景色画点，PRESET 用背景色画点。

3. 可以使用相对坐标。当使用 STEP 选项时，表示使用相对坐标。

例：画出在蓝色的天空中繁星密布的景象

【源程序】

```
'Example QB7—41.bas
SCREEN 1,0
COLOR,1
CLS
LOCATE 25, 10
PRINT "press any key to end"
DO
  X = INT(300 * RND)
  Y = INT(180 * RND)
  PSET (X, Y),3
LOOP UNTIL INKEY$ <> ""
END
```

7.9.4 画线或矩形

【格式】

LINE [STEP](x1,y1)-[STEP](x2,y2)[,彩色码][,B[F]][,线类型]

【功能】

在屏幕上画直线或矩形。

【说明】

参量 B——表示 LINE 语句要画一个矩形，其对角点坐标为(x1,y1)、(x2,y2)，

参量 BF——表示不仅要画出矩形，而且要用指定的彩色码填充矩形。

线类型——是一个 16 位的整数掩码。如选择该项时，LINE 从左到右读出每个位，如果一个位为 0，则不画该点，如果一个位为 1，则画出该点。因此，选择"线类型"参量可以画诸如虚线、实线及点划线等。该参量的取值范围为 0～&HFFFF，例如，当该参量为 &HFFFF(16 位全为 1)时，表示 16 个点都显示，画出实线，当它为 &HAAAA(-1010101010101010)时，每隔一点显示一点，画出虚线。当该参量省略时，画出实线，"线类型"对已填色的框无效。

例如：

```
LINE (0,0)-(319,199)          '在屏幕上画出一条对角线
LINE (160,100)-STEP(159,0)    '从中心点向右画一条水平线
LINE (50,50)-(170,170),,B     '画一矩形框
LINE (60,60)-(170,170),4,BF   '以红色画一矩形框，并用颜色填满。
```

例：利用 LINE 语句绘制正弦曲线

【源程序】

```
'Example QB7-42.bas
SCREEN 2
CLS
LINE (30, 99)-(310, 99)
PSET (30, 99)
FOR x = 0 TO 6.28 STEP .1
  y = 40 * SIN(x)
  LINE -(x * 50, 99 - y)
NEXT x
LOCATE 14, 1: PRINT "0"
LOCATE 14, 9: PRINT "90"
LOCATE 14, 18: PRINT "180"
LOCATE 14, 29: PRINT "270"
```

```
LOCATE 14, 40: PRINT "360"
END
```

7.9.5 画圆

【格式】

CIRCLE〔STEP〕(X,Y),〈半径〉,〔〈彩色码〉〕,〔〈起始角〉〕,〔〈结束角〉〕,〔〈纵横比〉〕

其中：

(X,Y)	为圆心的坐标，可为相对坐标；
〈半径〉	为指定圆的半径，为一整数值；
〈彩色码〉	指定线条颜色，缺省时为背景色；
〈起始角〉、〈结束角〉	为指定画圆弧的位置；
〈纵横比〉	即X轴半径与Y轴半径之比，用以调节画圆或椭圆的半径。

【功能】

以指定的圆心和半径画圆、圆弧或椭圆。

【说明】

1. (X,Y)圆心的坐标可为相对坐标STEP(X,Y)它是相对于当前光标的偏移量。

2. 起始角和结束角的单位为弧度。取值范围为$-2\pi\sim2\pi$，以逆时针方向进行度量，当角度为负时，表示圆弧的端点与圆心相连。如果起始角缺省，则其缺省值为0°。如果结束角缺省，则其缺省值为2π。

3. 纵横比是为了避免屏幕上纵向象元两点距离与横向象元两点距离的不等(在PC上，前者小于后者，即垂直像元点要比水平的少)造成圆变成椭圆的现象而设置的。当画圆时，该参数缺省，其缺省值为5/6(中分辨率)或5/12(高分辨率)，这样就起到了自动调整纵横比例的作用，使画出的图形为圆形。当纵横比小于1时，X轴是椭圆的主半径；当纵横比大于时1，Y轴是椭圆的主半径。

4. 除圆心坐标和半径外，其它参数均可缺省，但是在两个已选参数之间的缺省项，其逗号不能省。

5. CIRCLE语句完成绘图后，当前点是圆心。

例如：

```
CIRCLE (30,40),20   ' 圆心为(30,40)，半径为20的一个圆
CIRCLE (160,100),75,1.57080,4.71239   ' 圆心为(160,100)，半径为75的左半圆
CIRCLE (160,100),75,,,,1   ' 圆心为(160,100)，半径为75，Y轴为主轴的椭圆
```

例：先画两个椭圆，再画扇形和圆，最后，画一组逐渐增大的弧线

【源程序】

```
'Example QB7－43.bas
SCREEN 1
CIRCLE (60, 100), 80, 1, , , 3                    ' 画两个椭圆
```

```
CIRCLE (180, 100), 80, 2, , , 3 / 10
PRINT "PRESS ANY KEY TO CONTINUE"
DO
LOOP UNTIL INKEY$ <> ""
CONST pi = 3.141593
SCREEN 2
CLS
CIRCLE (320, 100), 200, , -pi, -pi / 2    '画扇形
CIRCLE STEP(-100, -42), 100               '画圆
PRINT "PRESS ANY KEY TO CONTINUE"
DO
LOOP UNTIL INKEY$ <> ""
CLS : sa = 0                              '画一组逐渐增大的弧线
FOR R% = 100 TO 200 STEP 10
    ea = (pi / 2.01)
    sa = pi / 4
    CIRCLE (320, 100), R%, , sa, ea
NEXT R%
END
```

7.9.6 图形的着色

【格式】

PAINT 〔Step〕(X,Y),〈区内着色〉,〈边界着色〉

其中：

〈区内着色〉　指定图形内所填的彩色码，可为整数或表达式，其值为 0～3，缺省时，在中分辨图率形模式下为 3。

〈边界着色〉　指定图形边界的彩色码，可为整数或表达式，其值为 0～3，缺省时，在中分辨率图形模式下为 3。

【功能】

在指定的坐标点所在的图形区域按指定颜色码着色。

【说明】

1. (X,Y)为边界线内的任意一点(称为内点)的坐标，它可以是绝对坐标形式，也可以是相对坐标形式。例如：

```
SCREEN 1:CLS
CIRCLE (160,100),50,1        '用青色画一圆
PAINT (160,100),2,1          '在圆内涂洋红色
```

2. PAINT 语句中给出的坐标，必须是在图形内的一点。否则将对图形以外的屏幕部分着色。如果该坐标恰好在图形的边框上，则不进行着色。

3. 需要着色的图形，必须是完全封闭的，否则，被涂的颜色将“漏出”到屏幕的其它地方。

4. 如果所着色的图形其内部与边部的颜色不同，就必须使用 PAINT 中的边界着色参数。如果该参数缺省，则边界与着色区的颜色一致。

5. 由于计算机遵循先画图后着色的作图原则，故该语句中边界着色参数应与先前作图语句中彩色码参数的选择值相一致。

例：绘制一个镖靶板。内圆是黑色的，而 3 个圆环分别是青色、洋红和白色

【源程序】

```
'Example QB7—44.bas
SCREEN 1: CLS
FOR n = 1 TO 4
  CIRCLE (160, 100), 10 * n, 3              '画四个圆
NEXT n
FOR n = 1 TO 3
  PAINT (175 + 10 * (n - 1), 100), n, 3   '对 3 个圆环着色
NEXT n
END
```

7.10 发声与音乐

7.10.1 发声语句

● 利用 PRINT 语句让电脑发声

ASCII 码为 7 的控制符能让电脑发出一声“嘟”的响声. 我们只要用 PRINT 语句输出这个代码，就可以听到声音。如：

```
PRINT CHR$(7)                 '计算机发出“嘟”声
PRINT CHR$(7)+CHR$(7)         '计算机发出“嘟嘟”声
DO
    PRINT CHR$(7)             '连续发出“嘟.....”声
LOOP UNTIL INKEY$<>""         '直到按键为止
```

● BEEP 语句

【格式】

```
BEEP
```

【功能】

使扬声器发声

【说明】

BEEP 语句发出与 PRINT CHR$(7)相同的声音。

例:在程序设计中经常需要用户回答是否继续某项工作,当要求继续时,按"Y",当要求中断时按"N",按其它键则发出"嘟"声,表示按键响应错误,要求用户重新按"Y"或"N",编程序模拟这一过程。

【源程序】

```
;Example QB7—45.bas
DO
  INPUT "Continue (Y/N)?", RESPONSE$
  Rs$ = UCASE$(MID$(RESPONSE$, 1, 1))
  IF Rs$ = "N" OR Rs$ = "Y" THEN
    EXIT DO
  ELSE
    BEEP
  END IF
LOOP
END
```

● SOUND 语句

【格式】

SOUND<频率>,<音长>

【功能】

通过扬声器发出指定频率和音长的声音。

【说明】

频率——周/秒,它必须是一个数字表达式,取 37～32767 范围内的整数。

当频率为 32767 时,喇叭不发声(即表示休止)。

音长——声音的持续时间,以 PC 机内部时钟的滴答(TICK)为单位,取 0～65535 之间的整数。

1 滴答=1/18.2 秒

例:利用 SOUND 语句模拟汽笛的声音

```
'Example QB7—46.bas
CLS
```

```
PRINT "sound is playing..... press ESC to quit"
DO
  FOR freq = 400 TO 815 STEP 15
    SOUND freq, 2
  NEXT freq
  FOR freq = 815 TO 400 STEP -15
    SOUND freq, 2
  NEXT freq
LOOP UNTIL INKEY$ = CHR$(27)
END
```

7.10.2　演奏音乐的 PLAY 语句

● PLAY 语句

【格式】

PLAY＜命令字符串＞

【功能】

演奏由给定的命令字符串所表示的音乐。

【说明】

在关键字 PLAY 后面的“命令字符串”是一种“曲调定义语言”，这种音乐语言由若干个单字符命令组成，可以表示音符的音高、音长及演奏方式等。

● 曲调定义语言

1. 音调的定义

有下面 2 种方法：

(1)音乐语言中的音调用字母 A—G 来表示。

以 C 调为例，音乐语言与简谱的对应关系如下：

简谱	1	2	3	4	5	6	7
	C	D	E	F	G	A	B

普通乐谱中的升半音和降半音用＃和 b 来表示；在音乐语言中，升高半音和降低半音分别用“＃”和“－”来表示，“＋”也可以表示升半音。在钢琴上，C—B 共 7 个音调使用的是白色键，而带升号或降号的音调则使用黑键。共计 12 个音调(因为 E—F 和 B—C 只升了半个音，故无 E^{+}和 $B^{\#}$等存在)。普通乐调中的 12 个音调与音乐语言的对应关系如表 7—7 和表 7—8 所示。

表 7—7　12 个音调的升半音表示

1	$1^{\#}$	2	$2^{\#}$	3	4	$4^{\#}$	5	$5^{\#}$	6	$6^{\#}$	7
C	C^{+}	D	D^{+}	E	F	F^{+}	G	G^{+}	A	A^{+}	B

表 7—8　12 个音调的降半音表示

1	2^{b}	2	3^{b}	3	4	5^{b}	5	6^{b}	6	7^{b}	7
C	D^{-}	D	E^{-}	E	F	G^{-}	G	A^{-}	A	B^{-}	B

例如:C 调的 12 个音调可以用下面的语句来演奏:

PLAY "CC+DD+EFF+GG+AA+B"

也可以写作:PLAY "CD—DE—EFG—GA—AB—B"

(2)用 Nn 命令来定义音符

同一个八度内共有 12 个音调,QB 支持 7 个八度,因此,总共有 84 个音调,用 Nn 命令可以演奏这 84 个音调中的任何一个音调,其中,n 的取值为 0—84,当 n 为 0 时,表示休止符。如:

N20　表示$\dot{5}$;　　N32　表示 5;　　N44　表示$\dot{5}$;

N21　表示$\dot{5}^{\#}$;　　N33　表示 $5^{\#}$;　　N45　表示$\dot{5}^{\#}$;

常见演奏音符见表 7—9。

表 7—9　常见音符的演奏值

演奏值	N20	N21	N22	N23	N24	N25	N26	N27	N28	N29	N30
音　符	$\dot{5}$	$\dot{5}^{\#}$	$\dot{6}$	$\dot{6}^{\#}$	$\dot{7}$	1	$1^{\#}$	2	$2^{\#}$	3	4
演奏值	N31	N32	N33	N34	N35	N36	N37	N38	N39	N40	
音　符	$4^{\#}$	5	$5^{\#}$	6	$6^{\#}$	7	$\dot{1}$	$\dot{1}^{\#}$	$\dot{2}$	$\dot{2}^{\#}$	

2. 音程的定义

On 命令

同一个音调中的音高从最低音到最高音依次为 C、D、E、F、G、A、B。如果要升高或降低一个或几个八度,则需要使用 On 命令,其中 n 的取值范围为 0—6,因此可以设置七个八度音程,每个音程用字母 O 和相应的数字来表示。

O2 表示普通低音音程

O3 表示普通中音音程

O4 表示普通高音音程

如: PLAY "O3CDEFGABC"　　'用普通中音演奏 7 个音调

如果要在音符 B 之后接着演奏一个高八度的 C，可以用下面的语句来实现：

PLAY"O3CDEFGABO4C"

用 On 命令可以设置 0 到 6 的任何一个八度音程。此外，如果只提高或降低一个八度，也可以用“>”或“<”来表示。

3. 音长(时值)的定义

Ln 命令

乐曲中的时值用 Ln 命令来定义，其中 n 的取值范围为 1－64，分别表示 1－1/64 个全音符，即：

L1　　全音符
L2　　半音符
L3　　三分之一音符
L4　　四分之一音符
…　　……
L64　　六十四分之一音符

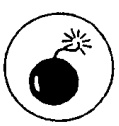
一个音符的时值也可以用音符后面加音长的方法来表示，其时值的计算与 L 命令相同。

例如，A4 与 L4A、C16 与 L16C 是等效的。但当连续多个音符的时值相同时，L 命令要方便得多。例如：

PLAY "O3C4D4E4C4D4E4F4G2E4F4G2" 可以简化成为：

PLAY"O3L4CDEC CDECEFG2 EFG2"

其中，L4 的含义为：如果在 L4 之后音符的后面没有数字，则该音符为四分之一音符。附点音符仍用点(.)来表示，一个点表示时值延长 50%。例如："A."为原长度的 3/2，"A.."为原长度的 9/4，而"A..."为原长度的 27/8。



当时值与升调或降调的音符连用时，升号(+)或降号(－)应写在时值之前。

4. 休止符的定义

Pn 命令

在 QB 中，用 Pn 命令表示休止符，其休止时间的计算方法与 Ln 命令相同。

例如：　P1 表示全休止符，休止 4 拍
　　P2 表示休止 2 拍
　　P4 表示休止 1 拍
　　P8 表示休止半拍

5. 演奏速度的定义

Tn 命令

QB 中用 Tn 命令来设置乐曲的节拍速度，n 的取值范围为 32—255，所表示的是每分钟内四分音符的数量。例如 T100 的节拍速度为每分钟 100 个四分音符。如果 Tn 命令缺省，则隐含为 T120。由于时钟中断速度慢，当演奏速度较高时（如在 T255 时的 L64），有些音符可能演奏不出来。

除 Tn 命令外，还可以用以下三个命令来控制演奏的速度，其中 MN 为正常，MS 为快速，ML 为慢速：

MN：设置"Music Normal"（乐曲正常），每个音符的演奏时间是 L 命令定义的 7/8。

ML：设置"Music Legato"（乐曲连奏），使每个音符的演奏时间完全由音长(L)设定。

MS：设置"Music Stacoato"（乐曲断奏），使每个音符的演奏时间是音长(L)的 3/4。

6. 演奏方式

演奏方式分为两种，即前台(MF)方式和后台(MB)方式。

前台方式(MF)　用这种方式演奏时，必须在前面的音符结束后，其后的音符才能开始。当演奏方式缺省时，QB 认为是前台方式。

后台方式(MB)　利用后台方式，可以在不影响正常工作的情况下演奏音乐。当以后台方式演奏时，通常把音符放入缓冲区中，缓冲区的最大容量为 32 个音符。如果想用后台方式演奏多于 32 个音符的乐曲，必须使用音乐陷阱。

7. 字符串命令 X

对于需要重复演奏的乐曲片断，可以先放在一个字符串变量中，然后再通过 X 命令反复引用，其功能与过程调用类似。例如：

S$ ="O3GG. F8EFGFE. D8CO2BGGO3C. O2B88O3CDEF"

在字符串 S$ 中存放着一段乐曲，可以用 X 命令来引用它：

PLAY "X"+VARPTR$(S$)

上述各命令(On,Nn,Ln,Pn,Tn)中的 n 可以是常数，也可以是变量。常数可以直接写在命令之后，如果是变量，则要在变量名的前面加一个等号。例如，假定节拍速度用变量 Tempo 来表示，则：

PLAY "T="+VARPTR$(Tempo)

例：

【源程序】

```
scale$ ="cdefgab"
Play "O0X"+VARPTR$(scale$)
for i=1 TO 6
    Play ">X"+VARPTR$(scale$)
next i
Play "O6x"+VARPTR$(scale$)
```

```
For i=1 TO 6
    Play "<X"+VARPTR$(scale$)
Next i
```

该例用">"使演奏从八度音程 0 到 6，然后再用"<"从八度音程 6 到 0 演奏。

【说明】

在实际编程中，所有的控制命令符，不一定每次每一个都要出现。

● 音乐命令字符串的描述方法

音乐命令字符串有两种描述方法：

1. 音程＋音调＋音长

例：利用"音程＋音调＋音长"方法演奏"两只老虎"

【源程序】

```
'Example QB7-47.bas
PLAY "mnt150l4o3"
PLAY "cdeccdec"
PLAY "efg2efg2"
PLAY "g8a8g8f8ecg8a8g8f8ec"
PLAY "co2go3c2co2go3c2"
END
```

2. 音长＋演奏值

例：利用"音长＋演奏值"方法演奏"两只老虎"

【源程序】

```
'Example QB7-48.bas
PLAY "mnt150"
PLAY "l4n25n27n29n25n25n27n29n25"
PLAY "n29n30l2n32 l4n29n30l2n32"
PLAY "l8n32n34n32n30 l4n29n25 l8n32n34n32n30l4n29n25"
PLAY "n25n20l2n25 l4n25n20l2n25"
END
```

附录 A

QB ASCII 表

Hex	Oct	Dec	ASCII	Hex	Oct	Dec	ASCII	Hex	Oct	Dec	ASCII
0	0	0	(nul)	20	40	32	(space)	40	100	64	@	60	140	96	`
1	1	1	☺ (soh)	21	41	33	!	41	101	65	A	61	141	97	a
2	2	2	● (stx)	22	42	34	"	42	102	66	B	62	142	98	b
3	3	3	▶ (etx)	23	43	35	#	43	103	67	C	63	143	99	c
4	4	4	◆ (eot)	24	44	36	$	44	104	68	D	64	144	100	d
5	5	5	♣ (enq)	25	45	37	%	45	105	69	E	65	145	101	e
6	6	6	◀ (ack)	26	46	38	&	46	106	70	F	66	146	102	f
7	7	7	• (bel)	27	47	39	'	47	107	71	G	67	147	103	g
8	10	8	◘ (bs)	28	50	40	(	48	110	72	H	68	150	104	h
9	11	9	(tab)	29	51	41	)	49	111	73	I	69	151	105	i
A	12	10	(lf)	2A	52	42	*	4A	112	74	J	6A	152	106	j
B	13	11	♂ (vt)	2B	53	43	+	4B	113	75	K	6B	153	107	k
C	14	12	♀ (ff)	2C	54	44	,	4C	114	76	L	6C	154	108	l
D	15	13	(cr)	2D	55	45	-	4D	115	77	M	6D	155	109	m
E	16	14	♫ (so)	2E	56	46	.	4E	116	78	N	6E	156	110	n
F	17	15	☼ (si)	2F	57	47	/	4F	117	79	O	6F	157	111	o
10	20	16	▲ (dle)	30	60	48	0	50	120	80	P	70	160	112	p
11	21	17	▼ (dc1)	31	61	49	1	51	121	81	Q	71	161	113	q
12	22	18	↕ (dc2)	32	62	50	2	52	122	82	R	72	162	114	r
13	23	19	‼ (dc3)	33	63	51	3	53	123	83	S	73	163	115	s
14	24	20	¶ (dc4)	34	64	52	4	54	124	84	T	74	164	116	t
15	25	21	§ (nak)	35	65	53	5	55	125	85	U	75	165	117	u
16	26	22	▬ (syn)	36	66	54	6	56	126	86	V	76	166	118	v
17	27	23	↨ (etb)	37	67	55	7	57	127	87	W	77	167	119	w
18	30	24	↑ (can)	38	70	56	8	58	130	88	X	78	170	120	x
19	31	25	↓ (em)	39	71	57	9	59	131	89	Y	79	171	121	y
1A	32	26	→ (eof)	3A	72	58	:	5A	132	90	Z	7A	172	122	z
1B	33	27	← (esc)	3B	73	59	;	5B	133	91	[	7B	173	123	{
1C	34	28	∟ (fs)	3C	74	60	<	5C	134	92	\	7C	174	124	|
1D	35	29	↔ (gs)	3D	75	61	=	5D	135	93	]	7D	175	125	}
1E	36	30	▲ (rs)	3E	76	62	>	5E	136	94	^	7E	176	126	~
1F	37	31	▼ (us)	3F	77	63	?	5F	137	95	_	7F	177	127	(Del)

Hex	Oct	Dec	ASCII	Hex	Oct	Dec	ASCII	Hex	Oct	Dec	ASCII
80	200	128	Ç	A0	240	160	á	C0	300	192	└	E0	340	224	α
81	201	129	ü	A1	241	161	í	C1	301	193	┴	E1	341	225	ß
82	202	130	é	A2	242	162	ó	C2	302	194	┬	E2	342	226	Γ
83	203	131	â	A3	243	163	ú	C3	303	195	├	E3	343	227	π
84	204	132	ä	A4	244	164	ñ	C4	304	196	─	E4	344	228	Σ
85	205	133	à	A5	245	165	Ñ	C5	305	197	┼	E5	345	229	σ
86	206	134	å	A6	246	166	ª	C6	306	198	╞	E6	346	230	µ
87	207	135	ç	A7	247	167	º	C7	307	199	╟	E7	347	231	τ
88	210	136	ê	A8	250	168	¿	C8	310	200	└	E8	350	232	Φ
89	211	137	ë	A9	251	169	⌐	C9	311	201	╔	E9	351	233	Θ
8A	212	138	è	AA	252	170	¬	CA	312	202	╩	EA	352	234	Ω
8B	213	139	ï	AB	253	171	½	CB	313	203	╦	EB	353	235	δ
8C	214	140	î	AC	254	172	¼	CC	314	204	╠	EC	354	236	∞
8D	215	141	ì	AD	255	173	¡	CD	315	205	═	ED	355	237	φ
8E	216	142	Ä	AE	256	174	«	CE	316	206	╬	EE	356	238	ε
8F	217	143	Å	AF	257	175	»	CF	317	207	╧	EF	357	239	∩
90	220	144	É	B0	260	176	▒	D0	320	208	╨	F0	360	240	≡
91	221	145	æ	B1	261	177	▓	D1	321	209	╤	F1	361	241	±
92	222	146	Æ	B2	262	178	│	D2	322	210	╥	F2	362	242	≥
93	223	147	ô	B3	263	179	┤	D3	323	211	╙	F3	363	243	≤
94	224	148	ö	B4	264	180	┤	D4	324	212	╘	F4	364	244	⌠
95	225	149	ò	B5	265	181	╡	D5	325	213	╒	F5	365	245	⌡
96	226	150	û	B6	266	182	╢	D6	326	214	╓	F6	366	246	÷
97	227	151	ù	B7	267	183	╖	D7	327	215	╫	F7	367	247	≈
98	230	152	ÿ	B8	270	184	╕	D8	330	216	╪	F8	370	248	°
99	231	153	Ö	B9	271	185	╣	D9	331	217	┘	F9	371	249	∙
9A	232	154	Ü	BA	272	186	║	DA	332	218	┌	FA	372	250	·
9B	233	155	¢	BB	273	187	╗	DB	333	219	▉	FB	373	251	√
9C	234	156	£	BC	274	188	╝	DC	334	220	▄	FC	374	252	n
9D	235	157	¥	BD	275	189	╜	DD	335	221	▌	FD	375	253	2
9E	236	158	₧	BE	276	190	╛	DE	336	222	▐	FE	376	254	■
9F	237	159	ƒ	BF	277	191	┐	DF	337	223	▀	FF	377	255	

附录 B

QB 出错信息

运行时的错误代码

代码	内　　容	代码	内　　容
2	syntax error	53	File not found
3	RETURN without GOSUB	54	Bad file mode
4	Out of DATA	55	File already open
5	Illegal function call	56	FIELD statement active
6	Overflow	57	Device I/O error
7	Out of memory	58	File already exists
9	Subscript out of range	59	Bad record length
10	Duplicate definition	61	Disk full
11	Division by zero	62	Input past end of file
13	Type mismatch	63	Bad record number
14	Out of string space	64	Bad file name
16	String formula too complex	67	Too many files
19	No RESUME	68	Device unavailable
20	RESUME without error	69	Communication—buffer overflow
24	Device timeout	70	Permission denied
25	Device fault	71	Disk not ready
27	Out of paper	72	Disk—media error
39	CASE ELSE expected	73	Advanced feature unavailable
40	Variable requirted	74	Rename across disks
50	FIELD overflow	75	Path/File access error
51	Internal error	76	Path not found
52	Bad file name or number		

有关错误的解释(按错误代码的顺序排列)

2. Syntax error

产生这个错误的原因很多,最常见的原因是编译时录入的 BASIC 关键字或参数出

错。例如,DATA语句格式不正确。

3. RETURN without GOSUB

RETURN语句之前找不到与之匹配的GOSUB语句。

4. Out of DATA

执行READ语句时,DATA语句的数据已读完。

5. Illegal function call

超界的参数传送到数字或字符串函数中,下列原因也会产生函数调用错误:

·使用负数或过大的下标。

·对负数进行非整数指数运算。

·使用GET或PUT文件时,记录数为负。

·赋予函数错误的或超界参数。

·对非磁盘驱动器进行BLOAD或BSAVE操作。

·在不支持I/O函数或语句的驱动器上进行该操作。如LOC或LDF。

·合并字符串时产生了比32767还要长的字符串。(运行态错)

6. Overflow

运算结果超界。其值超出了整型数范围及浮点数范围。

7. Out of memory

所需的内存大于能够提供的内存量。例如,分配文件缓冲区时空间不够。可以减少DOS缓冲区,或者清除一些终止并滞留在内存的程序或去掉一些设备驱动器。如果数组很大,需要在程序开始加入一条元命令。如果内存中装有一些文卷,清除文卷可以释放些内存空间。

9. Subscript out of range

数组元素下标超出了该数组定义的界限,或者存取一个没有维数的动态数组元素。在编译时如果使用调试(/D)选项,就会产生此报文。如果数组空间超过64KB,数组不是动态的,并且没有使用/AH选项,也会发生此错误,应该减少数组空间,使数组变成动态或使用/AH命令行选项。

10. Duplicate definition

对已被定义过的项使用定义语句。例如,在CONST语句和变量定义语句中使用相同的名字,或者过程与变量同名。

试图重复定义数组时,也会出现此错,重新定义动态数组时,必须使用DIM或REDIM。

11. Division by zero

表达式中除数为零,或指数运算结果为零的负数幂。

13. Type mismatch

变量的类型与要求不符合。例如,使用SWAP带有了字符串变量和数值型变量。

14. Out of string space

字符串变量超越了分配给字符串的空间。

16. String formula too complex

字符串太长或者 INPUT 语句中申请了 15 个以上字符串变量，应该将字符串公式拆开或将 INPUT 语句分成几部分使之正确执行。

19. No RESUME

程序在处理错误的过程中碰到程序结果标志。需要使用 RESUME 语句修正这种情况。

20. RESUME without error

在进入出错捕获子程序之前就遇到了 RESUM 语句。

24. Device timeout

在预定的时间内，程序没有从驱动器中接到信息。

25. Device fault

驱动器返回硬件出错，如果将数据传送给通讯文件时出现此报文，说明应该用 OPEN COM 语句测试的信号在这段时间内未被发现。

27. Out of paper

打印机无纸或未开电源。

39. CASE ELSE expected

在 SELECT CASE 语句中，没有匹配的情况。

40. Variable required

在以 BINARY 方式打开的文件上进行操作时，GET 或 PUT 语句不能定义变量。

50. FIELD overflow

FIELD 语句分配的字节大于随机文件中所说明的记录长度。

51. Internal error

QB 发生了内部故障，用带有你文卷的 Product Assistance Request 表，向 Microsoft 公司说明在什么条件下出现哪些报文。

52. Bad file name or number

语句或命令中使用的文件名或数字未经 OPEN 语句说明，或者超出了程序中说明的文件数目。

53. File not found

用 FILES，KILL，NAME，OPEN 或 RUN 语句对一个不存在的文件进行操作。

54. Bad file mode

出错原因如下：

- 程序对顺序文件使用 PUT 或 GET 语句，或者对非 I，O 或 R 文件模式使用 OPEN 语句。
- 对不是以随机存取格式打开的文件使用 FIELD 语句。
- 对以输入形式打开的文件进行输出操作。
- Quick BASIC 试图使用以压缩的格式存贮的包含文件，包含文件必须以文本格式存贮，重新装入包含文件，以文本格式存贮，然后再运行该程序。
- 试图装入一个错误的二进制程序。

55. File already open

从顺序输出方式的 OPEN 语句要打开一个已经打开的文件，或者用 UNK 语句对已经打开的文件进行操作。

56. FIELD statement active

文件 GET 或 PUT 语句中使用的记录变量空间已被 FIELD 语句占用，带有记录变量参数 GET 或 PUT 语句仅仅应用于没有 FIELD 语句的文件中。

57. Device I/O error

在驱动器进行 I/O 操作时出现 I/O 错误操作系统不能从错误状态恢复。

58. File already exists

NAME 语句定义的文件名已经以文件名的形式在磁盘上使用。

59. Bad record length

GET 或 PUT 语句中所说明的记录变量的长度与相应的 OPEN 语句中所说明的记录长度不匹配。

61. Disk full

没有足够的磁盘空间来完成 PRINT，WRITE 或 CLOSE 操作时；或者没有足够磁盘空间来完成向外设写目标文件或可执行文件时，会产生这个出错报文。

62. Input past end of file

使用 INPUT 语句读一个空文件，或者文件数据已经读完，可利用 EOF 函数测试文件结束来避免出现此错误。

63. Bad record number

在 PUT 或 GET 语句中，记录的个数小于等于零。

64. Bad file name

使用 LOAD、SAVE、KILL 或 OPEN 语句时，文件名非法(如：文件名字符态多)。

67. Too many files

编译时，包含文件嵌套层次大于 5 时，就会出现此种错误。运行时，试图用 SAVE 或 OPEN 语句建立新文件，使得文件目录数超过 255 个，也会发生此种错误。

68. Device unavailable

需要进入的设备没有联机或者不存在。

69. Communication—buffer overflow

在远程通讯中，接收缓冲区溢出，在 OPEN COM 语句中/C 选项或 RB 选项定义接收缓冲区的大小。应该经常查看该缓冲区空间或者经常将缓冲区置空(使用 INPUT 函数)

70. Permisson denied

对写保护的磁盘进行写入操作或读写加锁文件。

71. Disk not ready

驱动器的门未关闭，或驱动器上无磁盘。

72. Disk—media error

磁盘驱动器硬件在磁盘上查出了物理损伤。

73. Advanced feature unavailable

BASIC 子程序或函数中使用的参数个数不正确。

74. Rename acrossd disks

试图用新的驱动器名更改文件名，这是不允许的。

75. Path/File access error

在 OPEN，MKDIR，CHDIR 或 RMDIR 操作时，DOS 无法使路径名与文件名正确地联系起来。此操作不能完成。

76. Path not found

在 OPEN，MKDIR，CHDIR 或 RMDIR 操作时，DOS 无法找到指定路径。操作不能完成。

海淀走读 0028383

0035793

Document generated by Anna’s Archive around 2023-2024 as part of the DuXiu collection (https://annas-blog.org/duxiu-exclusive.html).

Images have been losslessly embedded. Information about the original file can be found in PDF attachments. Some stats (more in the PDF attachments):

{
  "filename": "MTAyNzUxNjguemlw",
  "filename_decoded": "10275168.zip",
  "filesize": 21966082,
  "md5": "d7aebe3c064d38cdec3b1c7fd92a3de8",
  "header_md5": "8329508c87175aeed3a7082ea47827cf",
  "sha1": "da87663f85af4f5d6878229a6a770d6fccfdadee",
  "sha256": "7156307f3e8bc5b6b401b2215f633953fdcd72715b2607570ebdf51a053664d1",
  "crc32": 2577685100,
  "zip_password": "28zrs",
  "uncompressed_size": 24507260,
  "pdg_dir_name": "10275168",
  "pdg_main_pages_found": 178,
  "pdg_main_pages_max": 178,
  "total_pages": 189,
  "total_pixels": 1284303074,
  "pdf_generation_missing_pages": false
}